

000046

**HP64000  
Logic Development  
System**

**Emulator/Analyzer  
6800/6802**



HEWLETT  
PACKARD

## **CERTIFICATION**

*Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.*

## **WARRANTY**

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its options, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service. Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

## **LIMITATION OF WARRANTY**

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## **EXCLUSIVE REMEDIES**

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

## **ASSISTANCE**

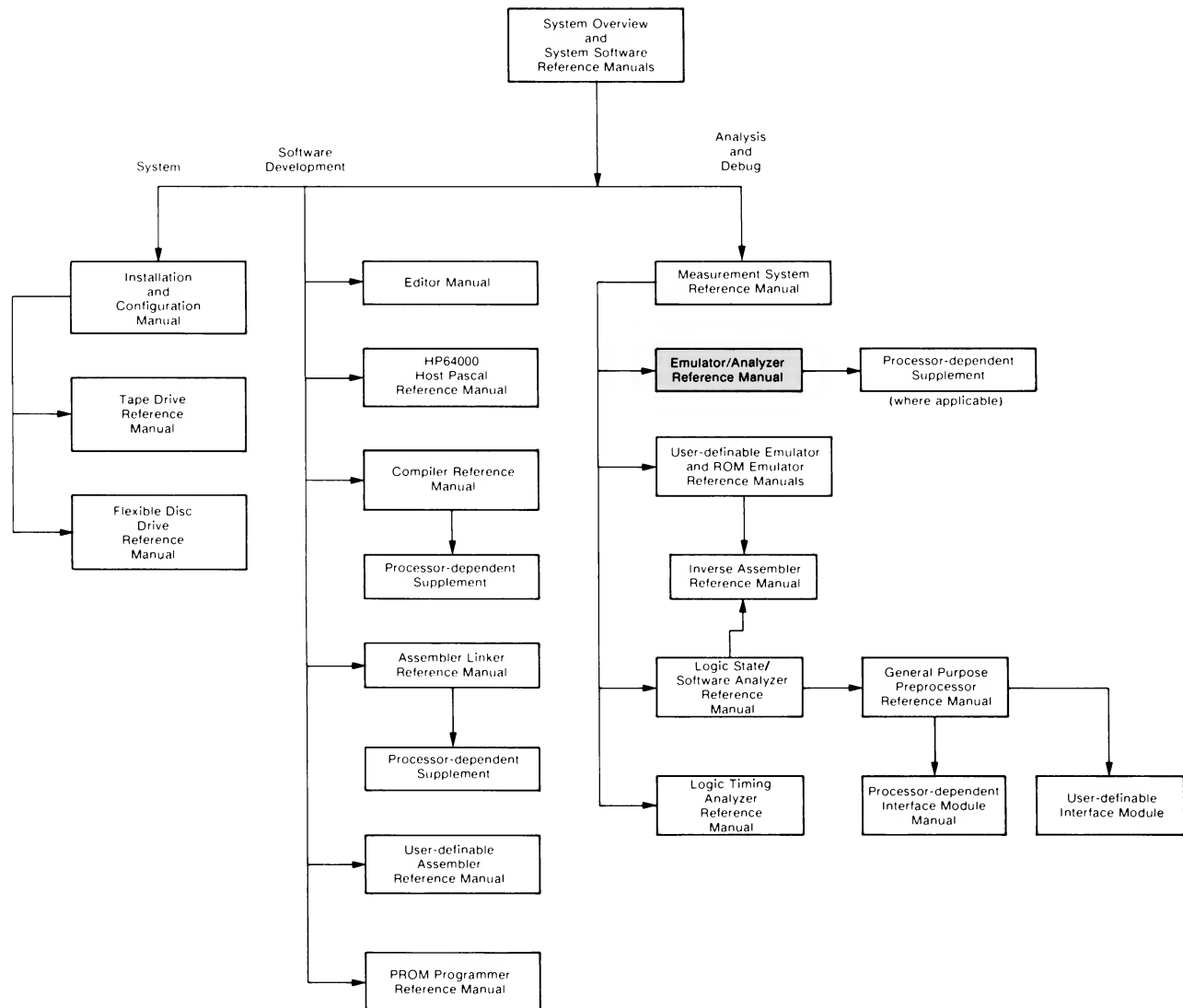
*Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.*

*For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.*

# Model 64000 Reference Manuals

The following block diagram shows the documentation scheme for the HP Model 64000 Logic Development System. The interconnecting arrows show the recommended progression through the manuals as a way of gaining familiarity with the system.

Manual Map



## Printing History

Each new edition of this manual incorporates all material updated since the previous edition. Manual change sheets are issued between editions, allowing you to correct or insert information in the current edition.

The part number on the back cover changes only when each new edition is published. Minor corrections or additions may be made as the manual is reprinted between editions.

First Printing.....	March 1980	(Part Number 64210-90902)
New Edition .....	March 1981	(Part Number 64210-90904)
Reprinted .....	March 1982	
Preliminary Edition.....	October 1982	(Part Number 64210-90905)
New Edition .....	December 1982	(Part Number 64210-90906)
Update U0283 .....	February 1983	

# Manual Conventions

The manual conventions and syntax conventions used in this book are presented below. For a full understanding of information in this manual, review the following conventions.

underlining

Where it is necessary to distinguish user input from computer output, the input is underlined.

[ \_ \_ ]

Dashed line key symbols indicate a softkey on the keyboard. The physical labels for the softkeys appear on the CRT display. In text, the softkey label will appear within the symbol.

RETURN

Solid outlined key symbols are used in text to represent labeled keys on the keyboard.

[      ]

Parameters enclosed in square brackets are optional. Several parameters stacked inside a set of brackets indicate an either/or situation. You may select any one or none of the parameters.

The use of square brackets implies that a default value exists.

## Example:

$\left[ \begin{array}{c} A \\ B \end{array} \right]$

This indicates A or B may be selected.

{      }

Braces specify that the parameter enclosed is required information. When several parameters are stacked within a set of braces, you must select one and only one of the parameters.

## Example:

$\left\{ \begin{array}{c} A \\ B \\ C \end{array} \right\}$

This example says one and only one of A, B, or C must be selected.

Choice of one and only one when several elements are enclosed.

## Manual Conventions (Cont'd)

[       ]  
[       ]       Stacked square brackets indicate that enclosed parameters are optional and may be selected in any single occurrence, any combination, or may be omitted.

### Example:

[ A ]  
[ B ]  
[ C ]

A and/or B and/or C may be selected, or this option may be omitted.

<       >       Angle brackets denote frequently used syntax elements which are predefined.

**lower-case  
bold type**       Key words (softkey commands) are lower-case in the Model 64000. These key words will always be represented in text with **lower-case bold type**.

### Example:

**edit** <FILE>

=>       Arrow indicates "is defined as."

. . .       An ellipsis indicates a previous bracketed element can be repeated.

UPPER-CASE       Literal information which is supplied in a command is represented in text with upper case type.

Syntax symbols in color       Indicates symbols are used for definition purposes and do not appear on the CRT display.

# **Emulator/Analyzer 6800/6802**

**©COPYRIGHT HEWLETT-PACKARD COMPANY/LOGIC SYSTEMS DIVISION 1982  
1900 GARDEN OF THE GODS ROAD, COLORADO SPRINGS, COLORADO, U.S.A.**

**ALL RIGHTS RESERVED**

# Table of Contents

## Chapter 1: Installation

Hardware Configuration .....	1-1
Installing the Emulation Pod and Emulation Control Board .....	1-3
Installing Emulation Probe To Target System .....	1-3
Installing the Analysis Board .....	1-5
Single Module Systems.....	1-5
Multiple Module Systems .....	1-5
Installing the Memory Control Board .....	1-5
Single Module Systems.....	1-5
Multiple Module Systems .....	1-6
Installing the Memory Boards.....	1-7
Single Module Systems.....	1-7
Multiple Module Systems .....	1-7
Installing the Bus Cables .....	1-8

## Chapter 2: Theory of Operation

Introduction .....	2-1
System Bus Structures .....	2-1
Emulation and Analysis Subsystem Functional Description .....	2-4
Subsystem Interfaces.....	2-4
Target System .....	2-4
Emulation Controller Functional Description .....	2-4
Transparency Considerations .....	2-5
Break Conditions.....	2-6
Emulation Processor Control .....	2-6
Emulation Memory Functional Description .....	2-9
Emulator Operating Modes .....	2-10
Internal Emulation.....	2-10
External Emulation .....	2-10
Running The Emulator .....	2-11
Emulation Memory and Target System Memory .....	2-11
Emulation Configuration .....	2-12
Using Symbols in Emulator Commands .....	2-12
Analyzer Characteristics .....	2-12
The Trace Command.....	2-13
Analyzer Status .....	2-13
The Display Command .....	2-13

## Chapter 3: Operating Fundamentals

Introduction .....	3-1
--------------------	-----

## Table of Contents (Cont'd)

Processor Architecture .....	3-1
Special Considerations - 6802 Microprocessor .....	3-2
Microprocessor Registers .....	3-4
Emulator Status .....	3-5
Numeric Status Specification .....	3-5
Softkey Status Specification .....	3-6
Operating Clock Specifications .....	3-6
6800 Emulator Clock Specifications .....	3-6
6802 Emulator Clock Specifications .....	3-7

### Chapter 4: Emulation and Configuration

Introduction .....	4-1
Assembly .....	4-1
Linking .....	4-1
Absolute File .....	4-2
Configuration .....	4-2
Measurement System Command Syntax .....	4-2
multiple module systems .....	4-3
measurement_system .....	4-3
em6800_S .....	4-4
single module systems .....	4-5
emulate .....	4-5
Execution .....	4-6
Running The Program .....	4-6
Configuration Questions .....	4-6
Card Selection .....	4-7
Clock Selection .....	4-8
Real-Time Mode Selection .....	4-8
Illegal Opcode Detection .....	4-9
Memory Configuration .....	4-9
Memory Map .....	4-10
Ending the Mapping Session .....	4-12
Simulated I/O Configuration .....	4-12
Interactive Measurement Configuration .....	4-14
Command File Designation .....	4-15

### Chapter 5: Operational Commands and System Command Files

Introduction .....	5-1
Command Line Comment Delimiter .....	5-1
Operational Command Syntax .....	5-1

## Table of Contents (Cont'd)

break .....	5-2
end .....	5-3
execute .....	5-4
halt .....	5-5
load .....	5-6
modify .....	5-7
modify configuration .....	5-9
modify memory .....	5-10
modify register .....	5-12
reset .....	5-13
run .....	5-14
specify .....	5-16
step .....	5-17
stop_trace .....	5-18
store .....	5-19
System Command Files .....	5-20
<CMDFILE> .....	5-20
Command Delays .....	5-21
wait .....	5-21

### Chapter 6: Display and List Commands

Display and List Command Capabilities .....	6-1
Memory Data .....	6-1
Register Contents .....	6-4
Trace Information .....	6-5
Global and Local Symbols .....	6-5
Display and List Command Syntax .....	6-6
display/list .....	6-7
display/list global_symbols .....	6-8
display/list loc_symb .....	6-9
display/list memory .....	6-10
display/list registers .....	6-12
display/list trace .....	6-13

### Chapter 7: Analysis and Interactive Commands

Introduction .....	7-1
trace .....	7-2
Status “and” Function .....	7-3
Using Analysis Commands .....	7-4
Interactive Measurement Selection .....	7-5

## Table of Contents (Cont'd)

### Chapter 8: Simulated I/O

Introduction .....	8-1
Overview .....	8-2
Common Attributes.....	8-2
Printer I/O Interface .....	8-3
Display I/O Interface .....	8-3
Keyboard I/O Interface .....	8-3
Disc Files I/O Interface.....	8-4
RS-232 I/O Interface .....	8-5
Printer I/O Interface.....	8-8
Open Printer (80H) .....	8-8
Write to Printer (82H) .....	8-9
Close Printer File (81H) .....	8-9
Display I/O Interface .....	8-11
Open Display File (80H) .....	8-11
Roll To/Write Line 18 (82H) .....	8-12
Select Starting Line/Column (83H).....	8-13
Write From Starting Line/Column (84H) .....	8-13
Close Display File (81H).....	8-13
Keyboard I/O Interface .....	8-17
User Program Requests Keyboard Read (80H) .....	8-17
64000 Response to Keyboard Read Request .....	8-18
64000 Detects Positive KB-Output-Command Word.....	8-18
User's Program Detects 00 in CA.....	8-19
Disc File I/O Interface.....	8-25
File Types .....	8-26
Creating New File .....	8-26
Creating File .....	8-26
Writing First Record.....	8-27
Writing Additonal Records.....	8-27
Closing Created File .....	8-28
Accessing Existing Files .....	8-28
Opening File .....	8-28
Selecting Record .....	8-28
Reading Record .....	8-29
Writing Record .....	8-30
Closing Open File.....	8-30
Deleting Files .....	8-30
Changing File Name Assigned to a Particular CA .....	8-30
RS-232 I/O Interface .....	8-38

## Table of Contents (Cont'd)

Open RS-232 File (80H) .....	8-38
Initialize 8251 (82H) .....	8-39
Asynchronous Mode.....	8-39
Synchronous Mode/Single Sync Character .....	8-40
Synchronous Mode/Double Sync Character .....	8-40
Command to 8251 (83H).....	8-40
Status From 8251 (84H) .....	8-41
Write To 8251 .....	8-41
Write Single Byte .....	8-41
Write Record, Update Write Buffer .....	8-41
Read From 8251 .....	8-43
Read Single Byte.....	8-43
Read Record, Update Read Buffer .....	8-43
Updating Read/Write Buffers (8DH) .....	8-44
Simulated I/O Error Codes .....	8-67
Simulated I/O Sample Programs .....	8-68
64000 File Formats .....	8-78
Assembler Symbols File (File Type 12).....	8-78
Record ID word .....	8-79
Symbol definition blocks .....	8-79
Checksum word .....	8-80
User Buffer/Assembler Symbols File Packing Formats .....	8-80
Linker Symbols File (File Type 13).....	8-80
Microprocessor Configuration Record .....	8-80
Global Symbols Records .....	8-82
Program Names Records .....	8-84
User Buffer/Linker Symbols File Packing Formats .....	8-85
Source File (File Type 2) .....	8-85
Listing File (File Type 5) .....	8-85
Absolute File (File Type 4) .....	8-85
First record .....	8-86
Additional records.....	8-86
Relocatable File (File Type 3).....	8-86
Program Description Record.....	8-87
Global Symbols Records .....	8-89
Data Records .....	8-89
External Symbols Records.....	8-90
Local Symbols Records .....	8-91
End Record .....	8-92
User Buffer/Relocatable File Packing Formats .....	8-92

## Table of Contents (Cont'd)

### Appendix A: Syntactical Variable Definitions

<ABSFILE> .....	A-1
<ADDRESS> .....	A-1
<ADR_LST> .....	A-1
<CMDFILE> .....	A-1
<FILE> .....	A-1
<REAL_VAL> .....	A-2
<STATE> .....	A-2
<VALUE> .....	A-2
<NUMBER> .....	A-3
<LOCAL SYMBOL> .....	A-3
<GLOBAL SYMBOL> .....	A-3
<MODULE> .....	A-3
<STRING> .....	A-3

### Appendix B: 6800/6802 Status and Error Messages

Status Messages .....	B-1
Error Messages .....	B-2

### Appendix C: Radio Frequency Interference

### Appendix D: Emulator Electrical Properties

### Index

## List of Illustrations

1-1. Installing the Emulation Probe .....	1-4
1-2. Memory Control Board .....	1-6
1-3. Address Range Selection Sockets .....	1-8
1-4. Memory, Emulation, and Intermodule Bus Cabling .....	1-9
2-1. 64000 Logic Development System Simplified Functional Block Diagram .....	2-3
2-2. Background Controller Transition Diagram .....	2-8

## List of Illustrations (Cont'd)

3-1.	6800 Emulator Pod .....	3-3
3-2.	6802 Emulator Pod .....	3-3
3-3.	Status Byte Format .....	3-5
6-1.	Memory Contents - Hexadecimal and ASCII .....	6-2
6-2.	Memory Contents - Mnemonic .....	6-3
6-3.	Register Contents .....	6-4
6-4.	Trace Memory Display .....	6-5
8-1.	Simulated Printer I/O Interface Diagram .....	8-5
8-2.	Simulated Display I/O Interface Diagram .....	8-6
8-3.	Simulated Keyboard I/O Interface Diagram .....	8-6
8-4.	Simulated Disc File I/O Interface Diagram .....	8-6
8-5.	Simulated RS-232 I/O Interface Diagram .....	8-7
8-6.	Display Techniques .....	8-16
8-7.	Keyboard I/O Interface Sequence .....	8-23
8-8.	8251 Initialization Formats .....	8-56
8-9.	Command Mode Instruction Format .....	8-57
8-10.	Asynchronous Mode Instruction Format .....	8-58
8-11.	Synchronous Mode Instruction Format .....	8-59
8-12.	8251 Status Word Format .....	8-60
8-13.	Writing RS-232 Record - Phase I .....	8-61
8-14.	Writing RS-232 Record - Phase II .....	8-62
8-15.	Writing RS-232 Record - Phase III .....	8-62
8-16.	Writing RS-232 Record - Phase IV .....	8-63
8-17.	Reading RS-232 Record - Phase I .....	8-64
8-18.	Reading RS-232 Record - Phase II .....	8-65
8-19.	Reading RS-232 Record - Phase III .....	8-66
8-20.	Reading RS-232 Record - Phase IV .....	8-66
8-21.	Simulated Display I/O - Sample Program A .....	8-68
8-22.	Simulated Display I/O - Sample Program B .....	8-71
8-23.	Simulated Keyboard, Display, and One Disc File I/O Sample Program .....	8-72
8-24.	Simulated Keyboard, Display, and Two Disc Files I/O Sample Program .....	8-75
8-25.	Assembler Symbol File Overall Structure .....	8-93
8-26.	Assembler Symbol Record Structure .....	8-94
8-27.	Assembler Symbol Record/User Buffer Format Details .....	8-95
8-28.	Assembler Symbol Record/Symbol Definition Block Examples .....	8-96
8-29.	Linker Symbol File Overall Structure .....	8-97

## List of Illustrations (Cont'd)

8-30.	Microprocessor Configuration Record Structure .....	8-98
8-31.	Global Symbol Record Structure .....	8-99
8-32.	Global Symbol Definition Block .....	8-100
8-33.	Program Name Record Structure .....	8-101
8-34.	Program Name and Address Definition Block Format .....	8-102
8-35.	Source and Listing Files - Overall Structure .....	8-103
8-36.	Source and Listing File Format .....	8-103
8-37.	Absolute File - Overall Structure .....	8-104
8-38.	Absolute File Formats .....	8-105
8-39.	Relocatable File Overall Format .....	8-107
8-40.	Relocatable File Program Description Definition Block .....	8-108
8-41.	Relocatable File Data Definition Block .....	8-109
8-42.	Relocatable File External Symbols Definition Block .....	8-110
8-43.	Relocatable File End Definition Block .....	8-111

## List of Tables

3-1.	Trace Status Softkeys .....	3-6
7-1.	“And” Function Results .....	7-4
8-1.	Printer I/O Codes .....	8-10
8-2.	Display I/O Codes .....	8-14
8-3.	Keyboard I/O Interface Codes .....	8-20
8-4.	Command Word Codes .....	8-22
8-5.	Disc File Type Numbers and Names .....	8-31
8-6.	Disc File I/O Codes .....	8-33
8-7.	RS-232 I/O Codes .....	8-45
8-8.	Simulated I/O Error Codes .....	8-67



# Chapter 1

## Installation

### Hardware Configuration

Information regarding the installation and configuration of emulation and analysis modules into the 64100A and 64110A systems, including power requirements and cabling, is found in the Installation and Configuration Reference Manual. Pay particular attention to power requirements when configuring multi-module systems.

Set the work station power switch to "off".

Unpack all emulation circuit boards, cables, pods and related equipment. Compare the parts received with the parts list to assure that all necessary items have been shipped. If any equipment is missing, contact the nearest Hewlett-Packard Sales/Service Office as soon as possible.

Carefully inspect the equipment for damage that may have occurred during shipping.

#### NOTE

---

The following installation steps assume the installation of a complete system (maximum memory). Particular attention should be paid to the power requirements for multi-module systems. Disregard procedure steps for equipment you have not purchased.

---

While the emulation and analysis circuit boards may be installed in any card slot in the station chassis, mechanical considerations make the following card groupings most convenient:

For single module systems:

<b>board</b>	<b>slot number 64100A</b>	<b>slot number 64110A</b>
Emulation Control board	9	0
Analysis board (optional)	8	1
Memory Control board (optional)	7	2
Memory board (optional)	6	3
Memory board (optional)	5	

For multi-module systems:

<b>board</b>	<b>slot number 64100A</b>	<b>slot number 64110A</b>
Memory board (optional)	9	
Memory board (optional)	8	
Memory Control board (optional)	7	
Emulation Control board	6	0
Internal Analysis board (optional)	5	1
Internal Analysis board (optional)	4	2
Emulation Control board	3	3
Memory Control board (optional)	2	
Memory board (optional)	1	
Memory board (optional)	0	

When an emulator is used in a system with state or timing analyzers, either half of the above ordering may be used.

Installation of the circuit boards is accomplished by aligning each circuit card in the circuit card guides, with the component side of the board facing forward, or up for the 64110A, and applying a gentle pressure until the board is seated in the mother board connector. Be sure the ejector handles are in their fully horizontal position.

# Installing the Emulation Pod and Emulation Control Board

For emulation of a 6800 microprocessor the model 64212A emulator pod is required. For emulation of a 6802 microprocessor the model 64213A emulator pod is required.

The 64211A Emulation Control board can be used with both the 6800 and the 6802 microprocessors.

Two multi-colored ribbon cables are used to connect the emulation pod to the emulation control board. One of the cables is connected to a surface-mounted connector, and one cable is connected to the top edge of the emulation control board. Pin 1 on the cable connectors is indicated by a triangle molded into each connector. Pin 1 of the board-mounted connectors is located at the left end of each connector. The surface-mounted connector is located near the top left corner of the Emulation Control board (on the component side). The edge connector is located at the left, near the surface mounted connector. Proper connection is facilitated by the color coding and keying of the connectors. Connect the pod to the control board by joining the connectors.

Install the Emulation Control board into the station chassis to maximize the free cable length outside the work station chassis, for single module systems.

## CAUTION

### PROTECT AGAINST STATIC DISCHARGE

---

The emulator pod contains devices that are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user plug to avoid emulator damage.

---

## Installing Emulation Probe To Target System

Carefully remove the target processor from its socket, and place the processor into a protected area. Then install the emulation probe into the vacant socket.

**CAUTION**

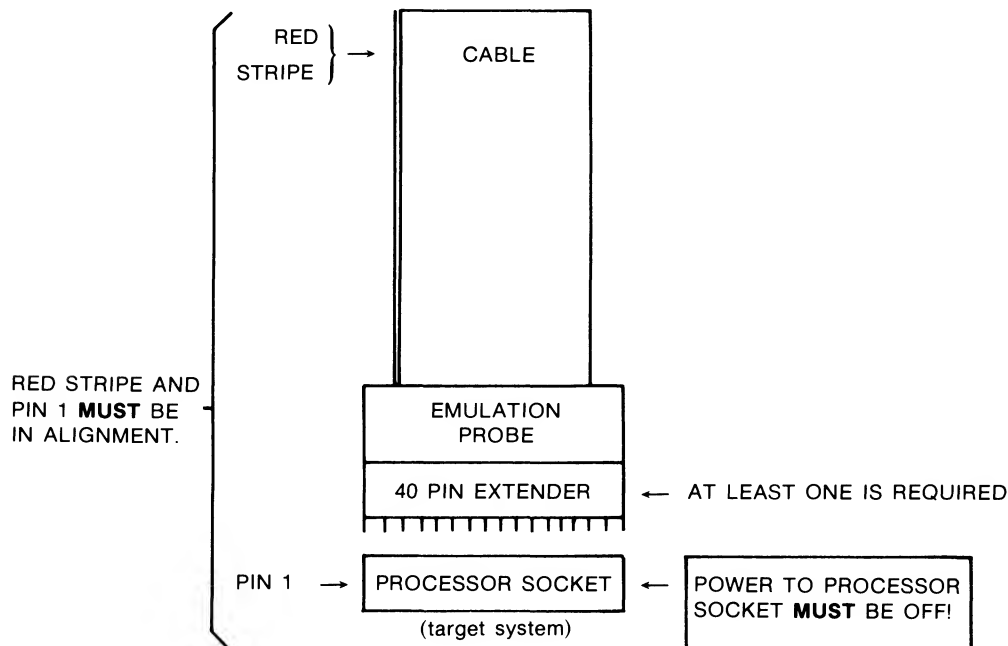
---

Do not install the emulation probe into the processor socket with power applied to the target system. The pod may be damaged if power is not removed before installation.

---

The emulation probe is provided with a pin protector that prevents damage to the probe when connecting and removing the probe from the microprocessor socket. DO NOT use the probe without a pin protector installed. If the emulation probe is being installed on a densely populated circuit board there may not be enough room to accommodate the plastic shoulders of the probe socket. If this occurs, another pin protector may be stacked onto the existing pin protector. The short wire extending from the emulation probe may be connected to the target system signal ground.

When installing the emulation probe, be sure the probe is inserted into the processor socket so that the red edge of the cable aligns with the pin 1 end of the processor socket as shown in Figure 1-1. Damage to the emulation equipment may result if the probe is incorrectly installed.



**Figure 1-1. Installing the Emulation Probe**

## Installing the Analysis Board

Either analysis board (64300A or 64301A) can be used with the 6800 or 6802 emulators.

### Single Module Systems

Install the Analysis board in the next slot adjacent to the Emulation Control board. For example, if the Emulation Control board was installed in slot 9, the Analysis board should be installed in slot 8. The board is installed with the component side facing the front of the work station. Avoid scuffing the emulation control cables when installing the Analysis board by ensuring that the cables are as flat as possible against the emulation control board.

### Multiple Module Systems

Install the internal analysis boards between the emulation control boards.

## Installing the Memory Control Board

The memory control board and memory boards are not required if only target system memory is to be used.

Set the data bits switch and address bus width cable to their correct positions before installing the memory control board.

The data bits switch, located between the J2 and J3 edge connectors, should be set to select a data bus of eight bits (switch moved to the left). Figure 1-2 shows the data bus switch in the correct position.

The address bus width is selected by positioning the ribbon cable, located at the board center, for a bus width of 16 bits. Figure 1-2 shows the cable position for a 16 bit address bus.

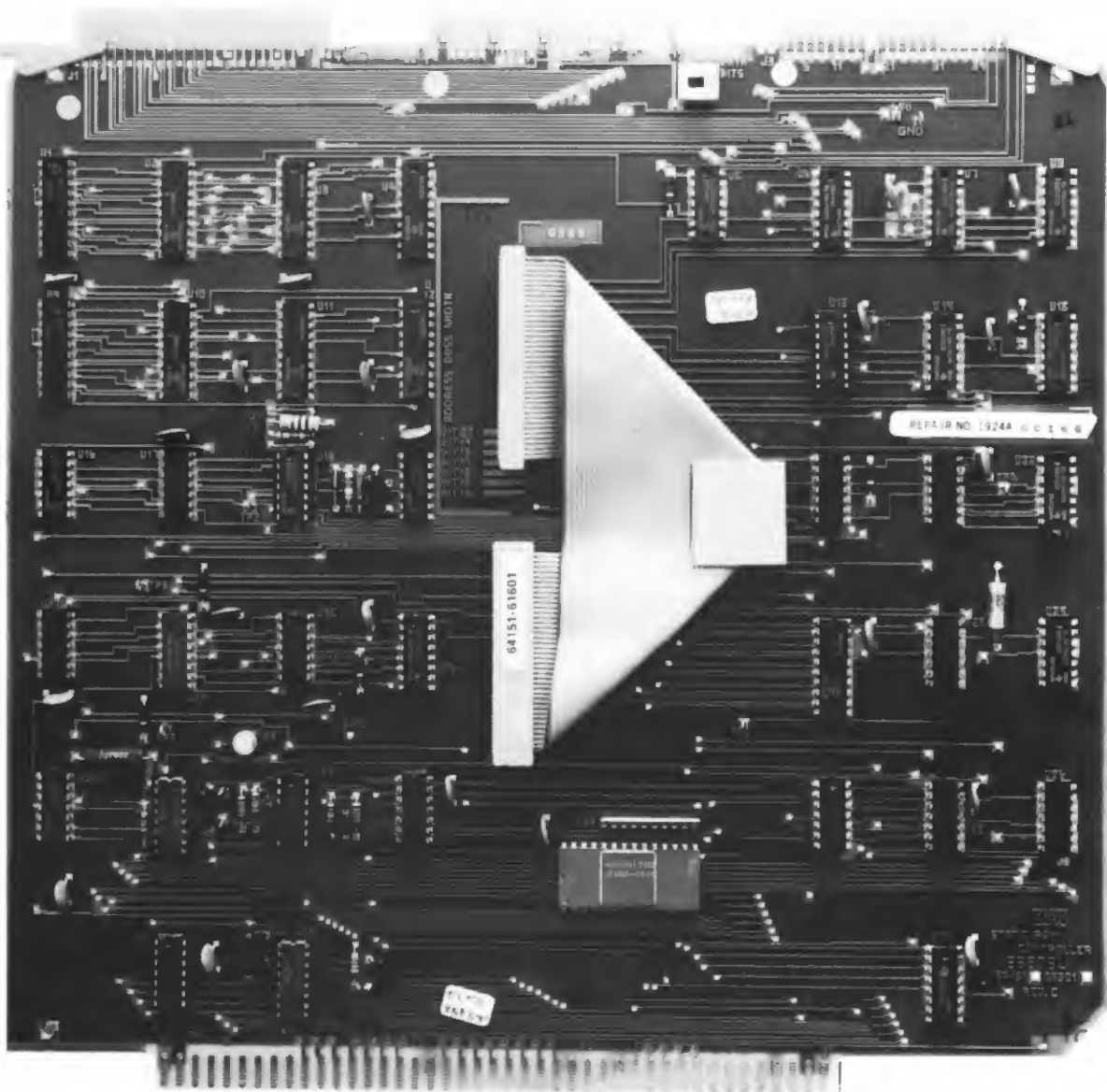
### Single Module Systems

The memory control board is installed in the next vacant slot, adjacent to the analysis board. If no analysis board is used, memory control may be installed in the slot next to the emulation control board, or, leaving a slot vacant where the analysis board would normally be placed, in a position two slots away from the emulation control board. As with the emulation control board, install the memory control board, with the component side forward, or up, with a gentle pressure until firmly seated in the mother board connector.

## Multiple Module Systems

The memory control boards should be installed outside of the emulation control boards, i.e., one should be in front of the front emulation control board and one should be behind the rear emulation control board.

There is room for only one memory controller in the 64110A card cage if a dual Emulation and dual Analysis configuration is installed.



**Figure 1-2. Memory Control Board**

# Installing the Memory Boards

Memory boards in the model 6415X series may contain from 4K to 16K words (8K to 32K bytes) of random access memory (RAM). The logical address of each board must be specified by installing an 8-pin jumper plug in socket U11. U11 is located near the upper right corner, and is adjacent to label boxes indicating address range options (see Figure 1-3). The address range is selected by installing the 8-pin jumper plug into the half of the socket (U11) associated with the address range being selected. The address range selection is used only to distinguish one memory board from another for the emulation and analysis system, and not to establish addresses for emulation. The address range may be split. For example, 0 to 16K and 48K to 64K may be found on the same board.

Memory boards in the model 6416X series may contain from 32K to 128K bytes of random access memory. Refer to the 6416X series service manual for the necessary configuration and installation procedures.

Address range specifications for the 6415X series memory boards are listed below:

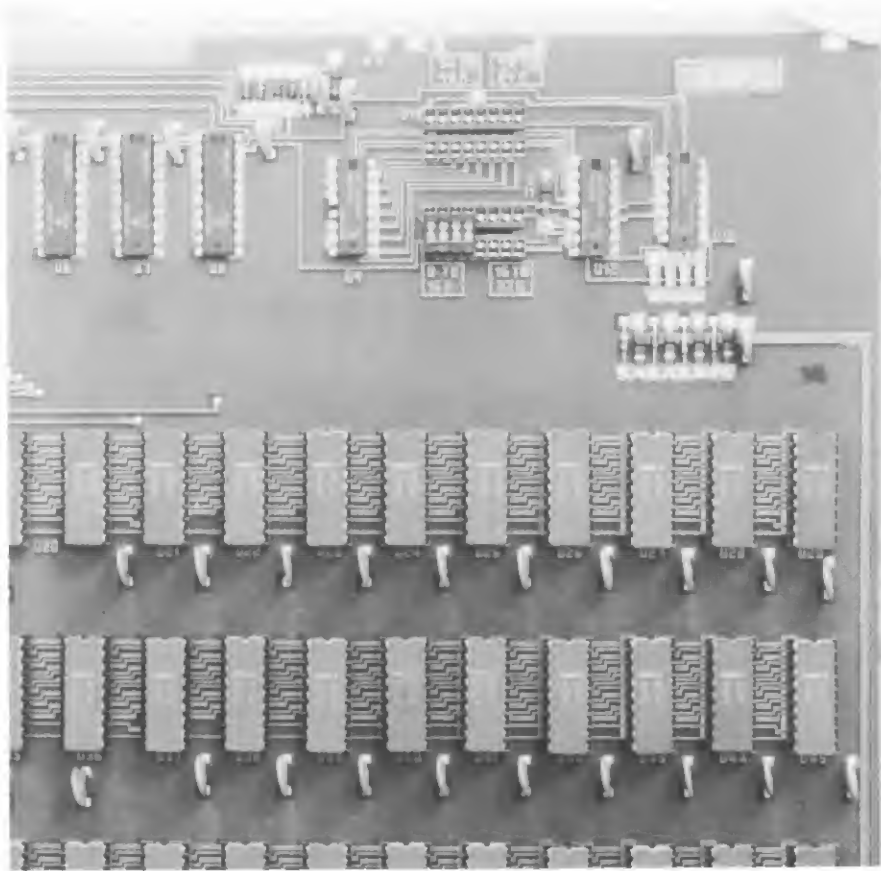
- a. Address range specification does not limit the address ranges which may be emulated.
- b. No two memory boards, connected to the same memory controller, may be specified as the same address range.
- c. Two positions are available for address range selection for 8 bit emulation. One full or partially loaded board must be set to 0 to 16K words address range.
- d. A maximum of two boards may be connected to a memory controller for 6800/6802 emulation processors. Refer to the Installation and Configuration Reference Manual for maximum allowable memory.

## Single Module Systems

Install the Memory boards in the slots adjacent to the Memory Control board. The boards are installed with the component side facing the front, or top, of the work station. Be sure that the boards are firmly seated in the mother board connector.

## Multiple Module Systems

Install the memory boards outside of the memory control boards. The memory boards should be the boards closest to the front and to the rear (bottom and top) of the card cage.



**Figure 1-3. Address Range Selection Sockets**

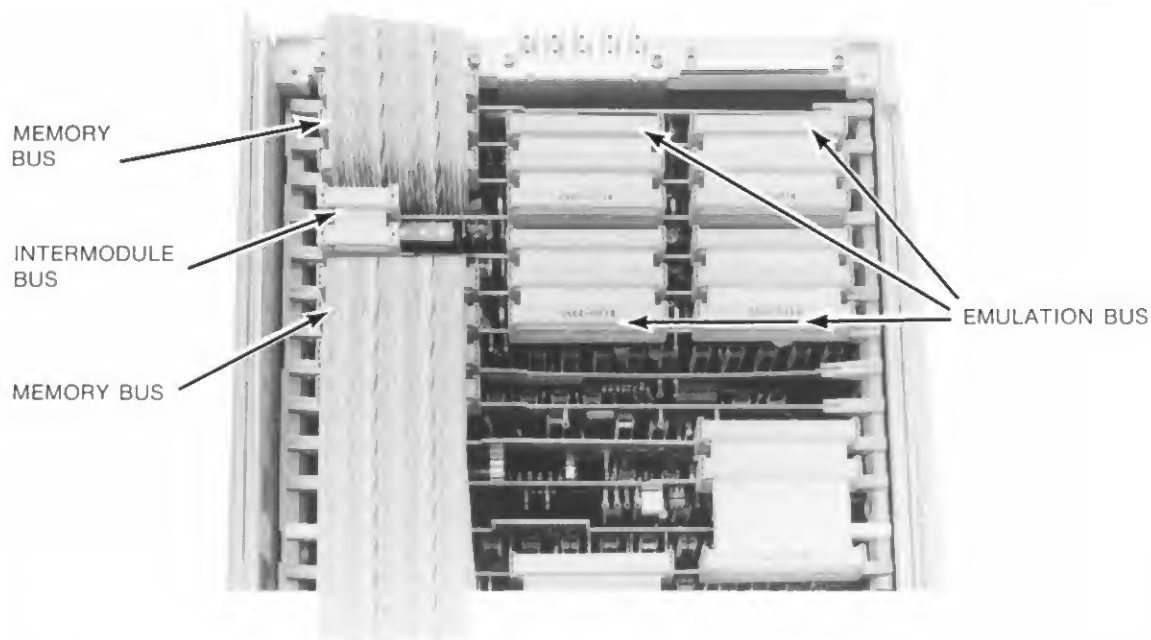
## **Installing the Bus Cables**

Bus cable installation should take place after installation of the circuit boards has been completed. See Figure 1-4 for the cable configuration for a complete system, including the intermodule bus if a multi-module system has been installed.

The two cables in the center and on the right of the circuit board set are the EMULATION bus cables. The connectors are keyed to facilitate correct installation. The connectors are also color coded, with the coding placed to the left end of each connector over pin 1. Each connector has a triangle indicator molded into the connector to indicate the location of pin 1 side and end in the connector. When properly installed, the red marker of the bus cable is on the left hand side of the cable when viewed from above the card cage. Two cables, each having three female connector blocks, are installed on the Emulation Control board, the Analysis board, and Memory Control board.

The Memory bus cable is on the left hand side of the board set, as you face the front of the work station. The Memory Control board is joined to the Memory boards by a cable with three connectors similar to the emulation bus cables. The connectors are color coded and keyed to facilitate proper installation, with the color coding placed to the left end of the connector. Each connector has a triangle indicator molded into the connector to indicate the location of pin 1 side and end in the connector. When properly installed, the red marker strip will be on the left of the ribbon cable as you look down on the card cage from the front of the work station.

The intermodule bus consists of a 20 conductor ribbon cable that is installed on the upper left corner of the appropriate board in each module. For emulation modules, connection is made to the internal analysis boards; for analyzer modules, connection is made to the analysis control boards.



**Figure 1-4. Memory, Emulation, and Intermodule Bus Cabling**

Figure 1-4 shows the cable placement in a 64100A card cage. The relative cable placement in a 64110A card cage is the same, although the card cage is rotated 90 degrees to the horizontal.



## Chapter 2

# Theory of Operation

## Introduction

The basic development system consists of a logic development station having a magnetic tape drive or flexible mini disc drive, an optional hard disc and printer, and software modules to edit, assemble or compile, link, and store program modules.

## System Bus Structures

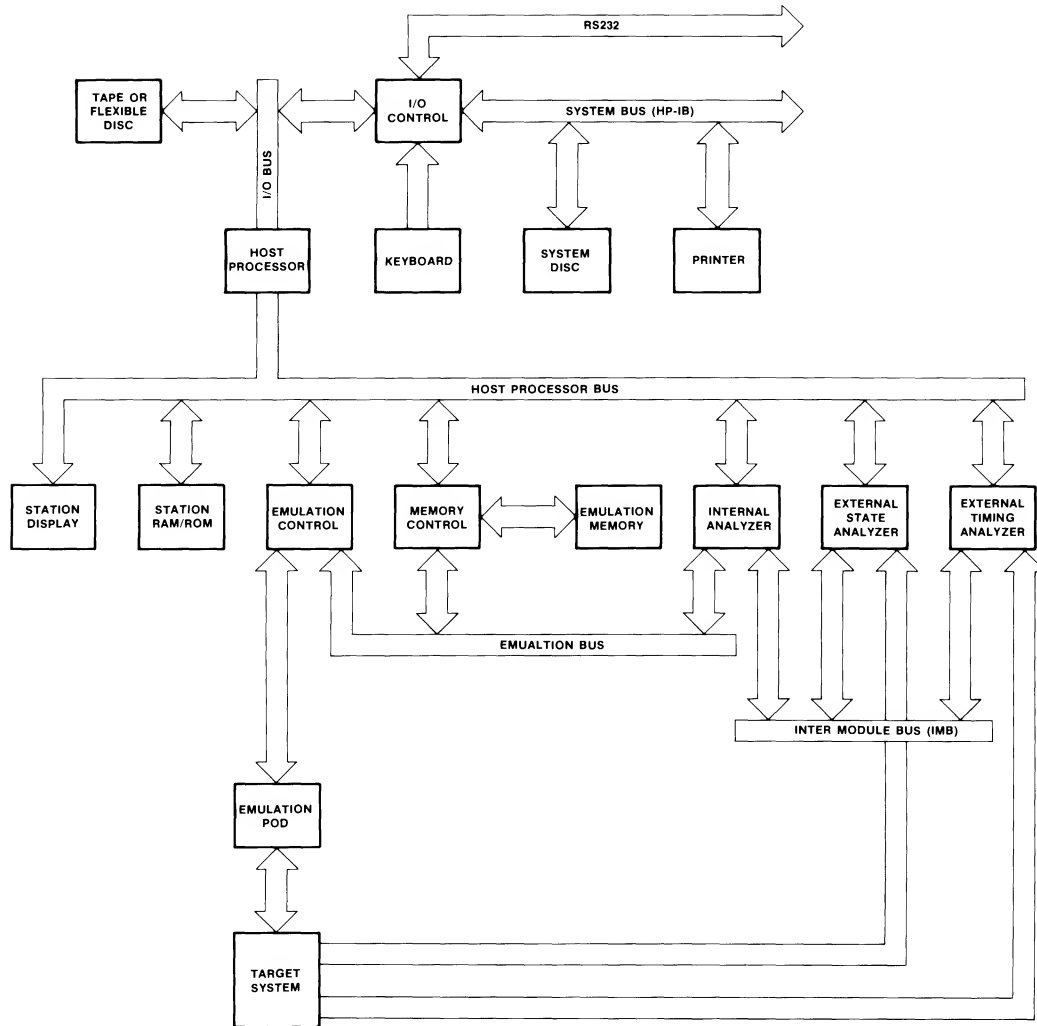
The 64000 system is designed with multiple independent buses for the host environment and emulator subsystem. Since the host processor and the emulation systems operate on separate buses, both can be running at the same time with no contention for system resources. Figure 2-1 illustrates the 64000 Logic Development Station bus orientation. The five basic bus structures for the 64000 are briefly described in the following paragraphs.

System Bus	The address, data, and control buses for the 64000 system are included in the system (HP-IB) bus. Communication between the printer, hard disc, and development stations occurs via the system bus.
Emulation Bus	The address, data, and control buses for the emulator processor are included in the emulation bus. Communication between the emulation controller, analysis module, and the target system takes place through the emulation bus.
Host Processor Bus	The host processor bus is the path through which the host processor communicates with the emulation and analysis subsystem, the display, and host processor memory.

I/O Bus	The Input/Output bus is dedicated to input and output devices of the 64000 station. It handles data to and from the minicartridge tape drive, the flexible disc drives, the keyboard, the hard disc drive, the printer, and the system processor.
Intermodule Bus	The intermodule bus connects the appropriate control boards in a multi-module system and carries signals related to sequence, timing, and triggering between the modules.

All data transfers in the emulation system occur on the buses described above.

For example, to display user RAM memory, a command is transmitted on the host processor bus from the host processor to the emulation control board. The Emulation Control board obtains the desired information through the emulation bus from the target system memory. The host processor then obtains the data from the emulation control board via the host processor bus. From there the data is passed to the display controller for display on the CRT. To display emulation memory, data in emulation memory is accessed by the host processor through the memory control board via the host processor bus. From that point on, data is transferred to the display.



**Figure 2-1. 64000 Logic Development System Simplified Functional Block Diagram**

The architecture of the multi-module system, illustrated in Figure 2-1, allows monitoring of the emulation processor without interfering with its operation. In addition, because the emulation bus is independent from the host processor bus, it is possible for emulation to continue while the development station is used for other purposes.

A major advantage of this architecture is the expandability of emulation systems. Since the host processing system does not restrict the word length or the speed of emulators connected to the host system, the system is capable of handling future as well as current microprocessors.

# **Emulation and Analysis Subsystem Functional Description**

A complete emulation and analysis subsystem consists of an emulation pod, emulation probe, emulation control board, memory control board, memory board(s), and analysis board. A brief description of the subsystem is given in the following paragraphs.

## **Subsystem Interfaces**

The interfaces for the emulation and analysis subsystem are the target system interface consisting of the emulation probe and pod, and the development station interface consisting of the host processor bus.

## **Target System**

The target system shown as part of Figure 2-1, represents a typical system having a microprocessor, control circuits, memory (ROM/RAM), and I/O circuits.

## **Emulation Controller Functional Description**

In foreground operation the emulation processor (in the emulator pod) functions as the processor for the target system. Programs executed by the emulation processor can be resident in the target system memory, or emulation memory, or a combination of both.

In addition, the target system memory can be loaded into emulation memory and the program modified. The memory map can be constructed to direct memory accesses to the emulation memory. The modified program can then be executed without disturbing the original version of the program.

During operation in the background state, emulation processor operation is suspended in the user system with the processor appearing to be inactive. This condition is implemented with the control of emulation pod buffers and latches by the background controller which is located on the emulation control board.

Operation of the emulator in the foreground state is exactly like operation of the target microprocessor in a normal environment.

Functional transparency of the emulator has been achieved with two features: background memory and the associated controller. The associated controller transfers processor control between the user program and the host system, i.e., foreground and background, respectively.

Background memory is located on the emulation control board. This memory is a 256 byte RAM which is accessible by the emulation processor and the 64000 host processor. The background memory is the primary communication link between the processors.

The background memory contains the routines for control of target processor execution. Routines to read and modify memory and registers and a routine to unload target processor registers are supplied by the host processor to the background memory. When the emulator changes the operating context of the emulation processor to background, the emulation processor will execute the routines in background memory.

A break to the background memory for the 6800/6802 emulators is accomplished by jamming a "Software Interrupt Instruction" to the emulation processor, i.e., forcing the processor to execute an SWI instruction that makes it dump its registers in known locations in BKG (background) memory, and to start executing BKG code.

## **Transparency Considerations**

A goal of emulation is that the emulation processor operates functionally and electrically in the same way as the target processor, i.e., to be transparent.

Functional transparency is achieved when an emulator places no restrictions or demands on any of the functional operations of the target processor, such as use of interrupts, restriction of memory address range, or any other functional characteristics.

Electrical transparency implies that all timing specifications, electrical loading, logic thresholds, drive levels, and any other electrical characteristics of the target processor are upheld by the emulator. The term "electrically identical" is a more accurate definition of electrical transparency.

Unfortunately, in attempting to achieve these goals, some compromises are sometimes necessary. Functional transparency cannot be achieved unless the "background activities" performed by the emulation system are shielded from the target system. These background activities include register interrogation, status checking, detection of illegal opcodes, or other operations that may disturb the operating context of the emulation processor.

The shielding or isolation of emulator background activities from the target system is accomplished with buffers and latches. These buffers and latches add propagation delays to the emulator which sometimes compromise electrical transparency.

The 64000 Logic Development System has been designed to implement functional transparency for the current generation of background control emulators. Users of the 64000 system, therefore, can do system design without arbitrary constraint from the emulators, being aware, however, of the slight propagation delays induced by the emulators.

## **Break Conditions**

A break condition initiates the context change of the emulation processor from foreground operation to background operation. There are four sources of a break condition: the logic analyzer, the emulation memory control board, the emulation control board, and the host system.

A break condition in normal operation is issued when an analyzer trace specification has been met, i.e., "break on trigger" or "break on measurement complete" is specified, or as a result of keyboard commands to the emulator that stop or single-step the emulation processor.

Detected errors account for break conditions from emulation memory and the emulation control board. Emulation memory control issues a break if an access to guarded memory occurs, or if a write to ROM occurs. A break condition from the emulation control board will be caused if an illegal opcode fetch occurs.

Other sources of break conditions occur during nonreal-time operation. Operations, such as register access and memory access, that occur during program execution will cause the alternation between foreground and background memory.

## **Emulation Processor Control**

The technique used by the 64000 emulator for emulation processor control involves jamming data information onto the processor data bus. This data jamming is asserted at the appropriate time in the processor instruction cycle to vector the processor operation to a control routine contained in the emulator background memory. The jamming process is synchronized by the background controller to occur on the first opcode fetch cycle following the occurrence of a break condition, thus allowing the emulator to gain control of the processor at the earliest possible time.

When the emulator has been changed to background state, the background program causes the register values of the processor, the program counter, and the next instruction address to be saved. This information is restored to the emulation processor when operation is returned to foreground (real-time) state. This allows the processor to continue execution from the point at which the break occurred when the emulator was in foreground. This process is similar to a hardware implementation of a non-maskable interrupt that is independent of the processor type.

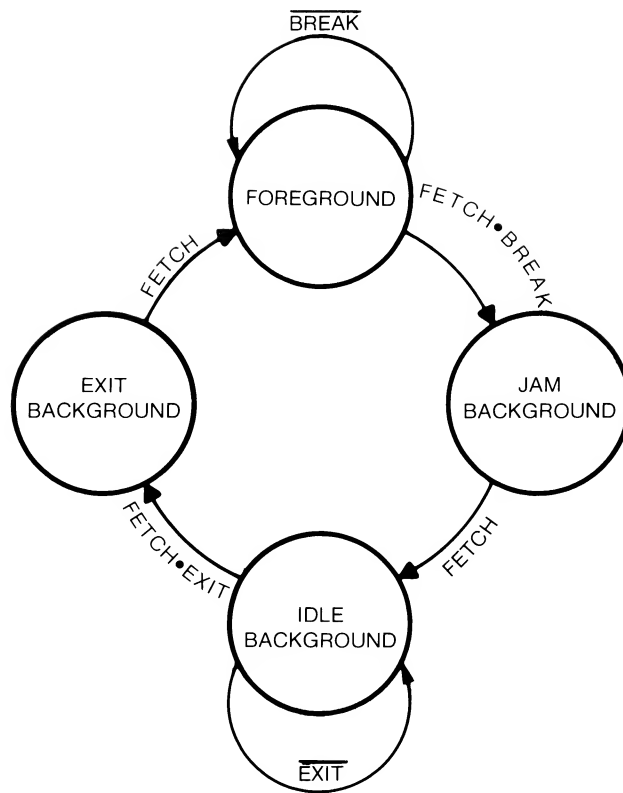
The background controller is a state machine consisting of four states: jam background, idle background, exit background, and foreground. Figure 2-2 is a diagram showing the background controller transition phases.

A state transition in the background controller will occur only at the beginning of an opcode fetch cycle that is coincident with other qualifying events. This is the earliest possible time after a break condition occurs, in which the jam state can occur.

Refer to the background controller transition diagram (Figure 2-2) for the following discussion. The background controller causes the emulation processor to enter the idle background state following the jamming operation. At this time, control of the address bus is returned to the emulation processor and the processor begins execution of a background entry program. During execution of the background routine, the processor registers are unloaded, return addresses are computed, and all other "housekeeping" tasks are completed to allow proper control for returning the emulation processor to foreground operation.

When these operations are complete, the emulation processor will enter a TRAP loop and wait for instructions from the host processor.

All host processor background memory accesses are totally transparent to the emulation processor. This makes it possible for the host processor to modify the jump address of the trap to coincide with the starting address of the background routine required to execute any host processor requests.



**Figure 2-2. Background Controller Transition Diagram**

The EXIT background state is entered when the host processor causes the emulation processor to make a jump to the EXIT routine in background memory.

In the EXIT routine, the emulation processor is directed to the desired foreground entry point at which instruction execution would have continued if a break condition had not occurred.

In the transition from the EXIT background state, the background controller enables the user interface buffers and allows processor execution of the foreground memory routine. Foreground memory can be user and/or emulation memory. Refer to the functional description of emulation memory for more information on foreground memory.

Transitions through the background controller states of Figure 2-2 occur whenever the host processor is used to control emulator operation or when any other break condition occurs. Single-stepping and continuous stepping of the emulation processor causes this type of transition through the background states.

# Emulation Memory Functional Description

Emulation memory for the 6800/6802 emulators consists of a memory control board and one or two memory boards. Each memory board can contain from 8K bytes to 32K bytes maximum of static RAM. Total emulation memory can be 64K bytes. This memory is for exclusive use by the emulator.

The memory controller provides the ability to map the target processor's address space into 1K byte blocks in the 64K byte address range.

This mapping function allows available target and/or emulation memory to be placed anywhere in the address range of the target processor. The memory controller also provides status bits to identify each block of memory, whether it is mapped or not. This allows the emulator to determine whether a block of memory is RAM, ROM, or undefined. If an illegal memory operation is attempted, such as a write operation to ROM, the memory controller will send a break signal to the emulator indicating an error condition.

The memory control board is the interface between emulation memory and the emulation control board. It is also the interface between the host processor and emulation memory. The host processor has no direct access to emulation memory. All memory accesses requested by the host processor go to the memory controller. The memory controller will grant a memory access to the host processor only if there is sufficient time between the emulation processor operations, or if the emulation processor is stopped. When the emulation processor is stopped, the memory controller will allow the host processor access to any memory location, including writes to memory mapped as ROM.

A write operation to emulation memory mapped as ROM is accomplished with the "modify" or "load" command.

The memory controller will not allow the emulation system to write to ROM since memory designated as ROM was defined in the context of the emulator.

Host processor "writes" to emulation memory are nonreal-time for the 6800/6802 emulators. Host processor "reads" of emulation memory take place in real-time mode.

# Emulator Operating Modes

The emulation system has two modes of operation: real-time, and nonreal-time. In addition, there are several options available through emulator configuration that affect these modes of operation. These options include the following:

- a. Detection of illegal opcodes
- b. Restricting emulation to real-time mode
- c. Memory mapping

The real-time emulation mode allows the user to run real-time emulation with or without a target system connected to the emulator. In addition, emulation memory and user memory can be used individually or in combination for real-time emulation. When emulating, consideration should be given to the emulation configuration and also to the intent of the emulation session.

## Internal Emulation

Internal emulation (no target system) is usually performed with the intent of debugging software. With internal emulation, the only clock that can be used is the internal clock of the emulator; therefore, code execution time will be relative to the internal clock speed. This should be kept in mind if the target system will have a different clock speed than the internal clock of the emulator.

## External Emulation

The 64000 can perform emulation in real-time or nonreal-time modes with or without a target system. If the real-time performance of the target system is important, emulation should be done in real-time with particular attention to the type of run commands and analysis commands issued during emulation.

In some cases emulation may be required to run in real-time mode because running in nonreal-time mode is not possible, such as with target systems that process interrupts and/or depend on a real-time clock for operation. Target systems of these types cannot be emulated thoroughly if real-time emulation is not available. Therefore, it is important to be aware of the types of emulator commands that will cause the emulator to operate in nonreal-time mode. Refer to the Real-Time Mode Selection portion of the configuration questions, in Chapter 4, for a list of those commands.

In addition, the user should realize what implications arise when emulation memory is used in part or in whole for emulation. The use of emulation memory could affect real-time emulation, depending on the implementation of the target system. Refer to the following paragraphs for use of emulation memory with respect to real-time emulation.

## **Running The Emulator**

There are other considerations that should be taken into account for running the emulator. The ability to detect illegal opcodes can be selected and the ability to restrict running of the target processor to a real-time mode can also be selected. These options are selected during configuration of the emulator. Refer to Chapter 4 for information concerning these features.

## **Emulation Memory and Target System Memory**

The use of emulation memory and/or target system memory can have some significance in the operation of the emulator. Ideally, emulation of a microprocessor should be done with as much of the final target system hardware as possible. Since this is not feasible at the beginning of the development cycle, the 64000 emulator provides emulation memory to replace target system memory during the project development stage.

In general, the final target system memory will not have the same specifications as the 64000 emulation memory. Therefore, selection of the clock for emulation can affect memory accesses by the target processor; so, the internal emulation clock would not be the clock to use with external hardware. This, in fact, is an illegal configuration for the emulation system. The emulator will, however, allow the user to specify use of the internal clock with a target system. When making a selection of a clock during configuration, there is the option to select an external clock to accommodate the target system.

## Emulation Configuration

Emulation software provides the interface between the emulator and the host processor in the development station. When the `emulate` softkey is pressed, the emulation software for configuring the emulator is loaded into the development station memory from the system disc. At this point, the display will present a series of questions for configuring the emulator to the user specifications. The options for configuration of the emulator are covered in detail in Chapter 4.

When the emulator has been configured, the program that the user wishes to execute on the target processor should be loaded to the emulator as discussed in Chapter 4.

## Using Symbols in Emulator Commands

Symbols may be used in any emulator command that allows expressions (as defined in Chapter 7). A symbol is always interpreted as the address value of that symbol. Variables in a program can be conveniently accessed by name. Even though it is legal to use a symbol as a data value in a trace command, remember that the symbol will be interpreted as the address value, not the data value stored at the referenced address location.

When using local symbols, the program module containing the symbol must be loaded by the emulator before the symbol can be used in a command. This is accomplished by using the “display” command or by specifying the program having the symbol with the format: `<SYMBOL>:<MODULE_NAME>`.

When using local symbols in emulator commands, only valid symbols will succeed as specifications. A list of qualified local symbols can be viewed by using the “display loc\_symb” command, or by referring to the `asmb_sym` file for the module.

The ability to use symbolic referencing in emulation provides a very convenient tool for debugging code which has been assembled or compiled on the 64000 system.

## Analyzer Characteristics

The 64000 system has an optional internal analysis board for analysis of emulation processor operation. The analysis capabilities are enhanced by the use of display or list commands, described in detail in Chapter 6.

## The Trace Command

The “trace” command can be specified with a wide range of complexity. In the simplest form, only “trace” need be specified. “Trace” also can be specified with a trigger, a qualifier, a count, a break, or combinations of any or all of those terms. In addition, the trace may be performed repetitively, in which program execution continues while the trace memory and trace display are updated; or the trace, with its most recent specification, can be performed by “trace again”.

The trace command causes program execution to be monitored and stored in chronological order in a 256 position trace memory. The trace memory can be displayed on the station CRT, or listed to a file or to the printer, for examination.

## Analyzer Status

Emulation analysis status can be specified with a numeric format from the keyboard, or through the softkey labels.

When status is specified with the numeric format, the specification may be in either hexadecimal, octal, or binary base. Status may also be specified using the four softkeys available. See Chapter 3 for details on status specification.

## The Display Command

An important feature of the 64000 emulators is the ability to display data for analysis in a format that is easy to interpret. This ability is implemented in the emulator by means of the “display” command. In addition to displaying the trace results, the “display” command allows the contents of memory, internal registers, and program symbols to be displayed. The display commands are described in detail in Chapter 6.

The “display count” mode selects either an absolute time of execution (elapsed time after the trigger), or relative time of execution (elapsed time between each state). See Chapter 7 for additional details about the “count” mode.



## Operating Fundamentals

### Introduction

This chapter contains general information pertaining to emulation and analysis of the 6800/6802 microprocessors. The information provided refers to aspects of the processor's architecture and status specification.

### Processor Architecture

The 6800 microprocessor is a memory-mapped I/O device with an 8-bit data bus and a 16-bit address bus. It contains six internal registers consisting of two 8-bit accumulators (A,B), a 16-bit index register (IX), six internal status flags (H,I,N,Z,V,C) located in an 8-bit condition code register (CC), a 16-bit program counter (PC), and a 16-bit stack pointer (SP).

The 8-bit data bus is a bi-directional bus. A stack, which must be located in RAM, is used to handle subroutine return addresses automatically during subroutine call and return instructions. All of the CPU registers (except the stack pointer) are automatically pushed onto the stack at the beginning of an interrupt service. A wait for interrupt (WAI) instruction provides quick interrupt servicing by placing all the CPU registers on the stack and then halting operation to wait for the interrupt. The 6800 has on chip Direct Memory Access (DMA) capabilities.

The 6802 microprocessor, in addition to the registers and accumulators of the 6800, has an internal clock oscillator and driver on the same chip. Additionally, the 6802 has 128 bytes of RAM located at addresses 0000H thru 007FH. The first 32 bytes of RAM, (addresses 0000H thru 001FH) may be retained in a low power mode by utilizing Vcc standby. The 6802 is completely software compatible with the 6800.

## Special Considerations - 6802 Microprocessor

The 6802 microprocessor pod has two external wires, separate from the user connector, labeled  $\overline{\text{INT VEC}}$  and  $\overline{\text{WROM}}$ . The purpose of the interrupt vector ( $\overline{\text{INT VEC}}$ ) wire is to allow the user to temporarily override interrupt vector fetches from emulation memory and fetch those vectors from outboard user memory. An example of the use of this line is with the Motorola 6828 Priority Interrupt Controller which alters interrupt vector addresses in the user system. Since these altered addresses cannot reach emulation memory, the user must supply the fetch locations in the target system. The  $\overline{\text{INT VEC}}$  input facilitates mapping this emulation space so that the user does not have to burn a new EPROM with each change to the rest of the memory in the EPROM space. The EPROM that the user does have in the target system need only contain the interrupt vectors located at the addresses used by the 6828 PIC chip for its fetches.

The  $\overline{\text{WROM}}$  wire is similar in nature in that it allows memory, mapped as emulation ROM, to be written to as an I/O type space. This requires that the user have address decoding in his target system such that anytime the ROM read/I/O write space is accessed the  $\overline{\text{WROM}}$  line is brought low. This line is qualified in the pod by R/W so that normal ROM read operations occur as usual. However, whenever that space is accessed as a write, the data is output to the user system instead of to the emulation memory where an illegal memory reference would occur (write to emulation ROM).

The 68000 emulation pod is shown in Figure 3-1; the 6802 emulation pod is shown in Figure 3-2.



**Figure 3-1. 6800 Emulator Pod**



**Figure 3-2. 6802 Emulator Pod**

# Microprocessor Registers

The microprocessors each contain six registers. The register names and functions are listed below. These names must be used when accessing the registers.

## Register

Name	Function
A	A accumulator - An 8-bit register used as a temporary holding register for ALU operations; may also be used as a general-purpose register.
B	B accumulator - Same function as A accumulator.
IX	Index register - A 16-bit register used to modify addresses in the indexed addressing mode. The IX register may be incremented, decremented, stored, loaded, or compared.
PC	Program counter - A 16-bit register which contains the next byte of the instructions to be fetched from memory.
SP	Stack pointer - A 16-bit register which contains the address where stack information is stored. The stack pointer is decremented (by one) immediately following the storage of a byte of information in the stack. For information retrieval, the stack pointer is incremented (by one) immediately before retrieving each byte of information from the stack.
CC	Condition code - An 8-bit register which contains status information used as test conditions for conditional branching.

The condition codes are:

- C carry/borrow
- V overflow
- Z zero
- N negative
- I Interrupt mask
- H half-carry

Polarity of the condition code flags when set = 1; when cleared = 0.

# Emulator Status

Emulation processor status can be specified to the analyzer in two ways: numerically or by softkeys. Specification of status must be in a format which the emulation processor recognizes. The status specification is used in the trace command in the following form:

status 0XXH

where “XX” represents the status byte.

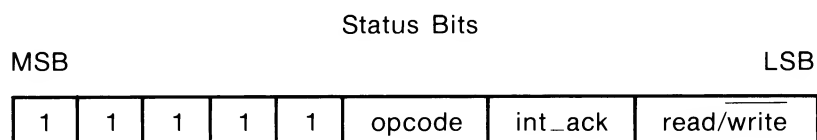
## Numeric Status Specification

An 8-bit byte is used to specify status numerically. The status byte may be specified in hexadecimal, octal, or binary, with “X” used for don’t care values. The status byte organization is shown in Figure 3-3.

For example, the specification:

trace only status 11111X01B

will cause a trace of read cycles, including opcode fetches.



opcode                      = 0 for first cycle of an instruction

int\_ack                     = 1 for interrupt acknowledge

read/write                 = 0 for write cycles  
                              = 1 for read cycles

**Figure 3-3. Status Byte Format**

## Softkey Status Specification

Trace specifications can be input using softkeys. That is, the appropriate bits are set for trace status qualification. An explanation of the trace status softkeys is given in Table 3-1. The “Softkey Label” column lists the name of the softkey, the “Binary Code” column lists the binary code making up the instruction, the “Command Line” column lists the command as it appears on the command line of the display when the softkey is pressed, and the “Remarks” column gives a brief explanation of the softkey function.

**Table 3-1. Trace Status Softkeys**

<b>Softkey Label</b>	<b>Binary Code</b>	<b>Command Line</b>	<b>Remarks</b>
opcode	0XXXXX001B	opcode	First cycle of an instruction
int_ack	0XXXXX111B	int_ack	Interrupt acknowledge
read	0XXXXX101B	read	Read memory
write	0XXXXX100B	write	Write memory

## Operating Clock Specifications

Proper operation of the 6800/6802 emulators requires that the operating clock speeds be within the specifications listed below.

### 6800 Emulator Clock Specifications

- All memory mapped - user: maximum clock frequency is 2 MHz, provided that  $T_{acc} < 235$  nanoseconds and is equal to peripheral read access time.
- Any memory mapped - emulation: minimum clock period is the greater of:

$$\begin{aligned} T_{cyc} &= 520 \text{ nanoseconds, or} \\ T_{cyc} &= 4t_{0r} + PW_{01H} + PW_{02H}. \end{aligned}$$

where:  $PW_{01H} < 180$  nanoseconds, and  
 $PW_{02H} > 295$  nanoseconds.

Note that  $PW_{0XH}$  = minimum high-level pulse width.

## **6802 Emulator Clock Specifications**

- a. All memory mapped - user: maximum clock frequency is 2 MHz, provided that  $T_{acc} < 245$  nanoseconds.
- b. Any memory mapped - emulation: maximum clock frequency is 1.4 MHz without wait states; 2 MHz with emulation wait states.

When driving the 6802 microprocessor pin 39 (EXTAL) with a TTL clock, pin 38 (XTAL) should be allowed to float high, i.e., left unconnected.



## Chapter 4

# Emulation and Configuration

## Introduction

In order to become familiar with the emulation and analysis user interface and feature set, it is recommended that a short program be written and executed with the emulation probe disconnected from the target system or “out of circuit”. A simple program that increments a single memory location or processor register will provide a good example.

## Assembly

In general, source files are generated using the 64000 editor. The first line of a program specifies the processor name in quotes followed by options on the same line. The assembler or compiler will generate the proper object code for the processor specified. The code generated will be placed in a file of the same name as the source, of type “reloc”. Also, a file of type “asmb\_sym” is generated. This file contains all of the symbols local to the module and their addresses. The address of a symbol may be absolute, or relocatable and relative to the program, data, or common program counter. This file is used to determine the addresses of local symbols used in emulation commands. If this file is not present during emulation, local symbols for that module cannot be referenced, displayed, or listed.

## Linking

Relocatable files must be linked together to create an absolute file. To begin the creation of a new absolute file, enter `link` followed by `RETURN`. This begins a sequence of questions which determine the files to be linked and their relocated addresses. The first question asks for object files. The name of the first program to be linked should be entered. Following the library files question the load addresses are requested.

## Absolute File

The last question to be answered when linking is for the name of the absolute file into which the relocated program is placed. The name given is also applied to a “link\_com” file, and a “link\_sym” file.

The “link\_com” file has in it the responses to the linker questions. The “link\_sym” file contains the names and addresses of all global symbols in the modules that have been linked; and contains the names and initial addresses of the PROG, DATA, and COMN program counters.

The program counter addresses are used to determine the addresses of all global symbols used in the emulation commands.

## Configuration

To begin emulation with the example program, the command is issued in the form “measurement\_system”, for multimodule systems, or in the form “emulate”, for single module systems. The syntax for each form is described later in this chapter. The command initiates a series of questions that configure the emulator for the particular application. Each question is provided with a default answer that can be entered as is with a **RETURN** or modified by using the softkeys or keyboard. The meaning of these questions and answers is described in detail later in this chapter. The questions and answers for interactive measurement are described in Chapter 7. For this example all of the default answers will be sufficient except during memory mapping. When the memory map is displayed, some portion of memory must be assigned. For this example the command “0 thru 0FFFH emulation ram” is sufficient provided the program has been loaded in this area of memory. The mapping section of configuration is exited with the “end” command. The last question asked during configuration is “Command file name?”. If a name is given, a file of type “emul\_com” will be created. This file is similar in function to the “link\_com” file and is described later in this chapter. For the example above a blank answer is sufficient but a file name may be entered.

## Measurement System Command Syntax

The measurement system can be entered by pressing either one of two softkeys. If more than one module is present in the card cage, the command **meas\_sys** will appear at the first level of softkeys. If an emulator is the only module present, then **emulate** will be present at the first level of softkeys.

For multiple module systems:

## measurement\_system

### SYNTAX

```
measurement_system [options continue]
```

### DEFAULT VALUE

measurement\_system is treated as a new entry into emulation

### FUNCTION

The command “measurement\_system” causes system operation to enter the measurement system monitor. The measurement system monitor coordinates and displays the interaction between the modules present and, in multiple module systems, controls entry to and exit from the individual modules of the system. Once in the monitor program, the emulator can be entered by issuing the command “em6800\_S” where “S” is the slot number of the emulation control board. The choice is made through the softkeys.

The “continue” option allows reentry to a previous session without disrupting a measurement in progress. If “continue” is not specified, all measurement system modules will be reset to their default configuration and any activity stopped. A “continue” is not possible under any of the following conditions:

- a. Power has been cycled or the station reset by shift/reset.
- b. Performance Verification (option\_test) has been initiated.
- c. The last session was exited by reset/reset.
- d. The measurement system configuration file is not present.
- e. A module was exited in a noncontinuable manner.

# em6800\_S

## SYNTAX

em6800\_S [<CMD\_FILE>]

where "S" is the slot number of the emulator control board, and <CMD\_FILE> is an optional emulation command file.

## DEFAULT VALUE

<CMD\_FILE> The last specified command file.

## FUNCTION

The "em6800\_S" command, when issued from the measurement system monitor program, transfers operating control to the emulator. If no command file is specified, or there is a conflict between the specified command file and the current hardware configuration, the emulation configuration questions will be initiated. A new command file will be generated, or the specified file will be edited.

For single module systems:

# emulate

## SYNTAX

```
emulate [<CMDFILE>] [load <ABSFILE>] [options { edit  
continue } ]
```

### Examples:

```
emulate  
emulate LOOP  
emulate LOOP load MUCH
```

## FUNCTION

If no options are selected, emulation configuration is initiated and a new command file is constructed.

## PARAMETERS

<CMD_FILE>	If <CMD_FILE> is specified, an emulation session is initiated using the configuration specified by the command file. If the current hardware configuration is inconsistent with the specified command file, editing of the command file configuration questions is forced.
[load <ABSFILE>]	The absolute file named will be loaded into memory upon entry of the emulation session.
[options edit]	The configuration questions will be presented with the specified command file supplying the default answers. If no command file is given, the result will be the same as if “emulate” had been selected.
[options continue]	The emulation session will be entered without changing the state of the emulator.

## Execution

After configuration the execution portion of emulation is entered. In this case the processor has been reset and is running in background. This condition is reported on the Status line of the display (Status: 6800/2 Reset in background). At this point the absolute file must be loaded into emulation memory using the load command in the form "load <ABSFILE>".

## Running The Program

Once the example program has been loaded, the run command can be issued to begin execution of the program. If the command "run" is given, program execution will begin at the transfer address specified in the source program. This is either the label given with the END pseudo-op at the end of an assembly language module, or the main routine of a PASCAL program. Thereafter "run" will cause execution to begin at the next program counter address as specified in the register display. If "run from <ADDRESS>" is issued, execution begins at the address specified.

## Configuration Questions

The emulation configuration questions are used to prepare the emulation hardware and software for a specific application. Each question is displayed along with a default response, with additional options shown in parentheses. Selecting the default responses will set up the emulation configuration that is easiest to use in most applications. The default response can be selected by pressing the **RETURN** key; or another response can be selected by the appropriate softkey, or by typing in a suitable response.

Once the questions have been answered for the particular application, the answers can be stored in a command file on disc so that the question and answer sequence need not be repeated for each emulation session. If changes to an emulation command file are desired, the file can be edited using the **modify** **config** softkeys. This allows changing only specified answers. At the end of the modify configuration sequence, a new file name can be assigned to the edited configuration, or the old file can be over written with the new information.

Throughout this discussion, the available softkey entries for each question are listed following the question. If an emulation command file is being edited to reconfigure the emulator, the default responses provided are the responses that were entered when the command file was originated or last edited.

The questions are divided into the eight sections listed below.

- a. Card Selection
- b. Clock Selection
- c. Real Time Mode Selection
- d. Illegal Opcode Detection
- e. Memory Configuration
- f. Simulated I/O Configuration
- g. Interactive Measurement Configuration
- h. Command File Designation

These sections are discussed on the following pages. The questions discussed in the first section are only presented when more than one emulation control board and/or more than one memory control board is installed in the 64000.

## Card Selection

It may be necessary, in multiple module systems, to specify the slots of the memory controller and internal analysis cards associated with the emulator being used. The following questions will appear:

Slot number of memory controller card?    0..9    (none)

The default answer will be the slot number of one of the memory controllers, or the slot number specified in a command file. It is possible for emulation to take place without a memory controller, provided that all memory is user memory. The ability to detect illegal memory references while the processor is executing the target program cannot be provided without the memory controller installed.

Slot number of analysis card?    0..9    (none)

The default answer will be the slot number of one of the analysis cards, or the slot number specified in a command file. It is possible to emulate without the benefit of an analysis card by selecting "none". None of the functions, however, that require an analysis card will be usable. The functions requiring an analysis card are "run until", and "trace".

## Clock Selection

Microprocessor clock source?    internal    (external)

internal                      Selects a 1 MHz clock source in the probe pod; this source should be selected when operating without a target system.

external                      Selects the clock source in the user system.

## Real-Time Mode Selection

The question listed below provides an opportunity to restrict the emulator to real-time program execution. "Real-time" in this case is not based on whether wait states are inserted or not, since none are needed by the emulator. Instead, real-time refers to the continuous execution of the user's program without interference from the development system except as instructed by the operator.

Interference can come from two sources: 1) stopping the processor (and DMA activity) so that the host processor can access memory, 2) automatically breaking into the background memory. Host processor access to emulation memory usually stops the emulation processor for 34 to 40 microseconds. The display/modify emulation memory features access emulation memory and pause the emulator once for every location that is specified. Features that utilize the background memory are display/modify registers and display/modify memory.

Features that cannot be performed in real-time mode are the following:

- a. memory accesses - Display, list, load, modify, and store.
- b. register accesses - Display, list, and modify.
- c. simulated I/O.
- d. symbol accesses - Display and list. These commands will be performed with the contents field showing "\*\*\*".

Breaking into the emulation background memory will happen if a feature that requires the background memory is invoked while the processor is executing user programs. After the feature is completed the processor is returned to the user program.

Restrict to real-time runs? no (yes)

no If runs are not restricted to real-time, all keyboard commands will be accepted.

The host processor will generate a break into the background memory if a feature is invoked which requires the background memory and the processor is executing a user program.

yes If operation is restricted to real-time runs, emulator features like display/modify memory and display/modify registers, which require the host processor to access emulation memory or utilize the background memory, must be enabled by an explicit break. Breaks can be generated by an analysis (“trace break\_on...”) command, by the emulation memory controller (access to illegal memory or write to ROM), or from the keyboard by entering “break”.

Features that require a break (with the exception of the load command) are disabled by the “run” command, and no automatic breaks into the background memory will be performed. The user’s system will not experience any pauses or other interference once the “run” command has been entered.

## Illegal Opcode Detection

Break processor on illegal opcodes? yes (no)

This option helps find unexpected executions in absolute code. If “yes” is selected, the processor will stop emulation if an invalid opcode is fetched. If “no” is selected, the emulation processor will attempt to execute the opcode in the same manner as the microprocessor unit being emulated.

## Memory Configuration

In order to perform emulation, the memory mapper must be properly programmed to correspond to the desired emulation and user system memory resources. The memory mapper allows the user to divide the processor’s address space into a number of blocks that can be individually assigned any one of five descriptors: emulation RAM, emulation ROM, user RAM, user ROM, or guarded memory. During emulation, the mapper monitors the address bus and provides the descriptor for the address present at any given time. This information is used by the emulator hardware to control the flow of data between the emulation processor and the memory resources.

The responses to the memory configuration questions are used to configure the memory mapper. These questions are explained on the following pages.

Modify memory configuration?    no    (yes)

### Memory Map

The memory map describes the partitioning of the processor address range into emulation RAM/ROM, user RAM/ROM, or illegal space for the emulation system.

The map is composed of from 0 to 32 entries plus memory default. Each entry must be an integral multiple of the 1K byte (1024 bytes) block size. The remaining parts of the address range, not covered by an entry, are mapped to the memory default.

At the top of the map display is information describing the amount of emulation memory available to be mapped, the amount already mapped, and the memory block size. The remainder of the screen above the status line shows up to 32 entries arranged in two columns. Each entry is displayed as an entry number, the address range covered, the memory type, and then (if the type was emulation memory) the physical block number actually used.

Entries are made by typing in a single address for a single block, or an address range followed by either:

(a) user     $\left\{ \begin{array}{c} \text{rom} \\ \text{ram} \end{array} \right\}$

(b) emulation     $\left\{ \begin{array}{c} \text{rom} \\ \text{ram} \end{array} \right\}$  [overlay <ADDRESS>]

(c) guarded

If a single address was specified, the entire 1K byte block containing that address will be mapped, e.g., 55H emulation ROM will map the range 0-3FFH to emulation ROM.

where:

rom - designates memory for which the memory control circuitry can detect the occurrence of write cycles and, if it is also emulation, memory which cannot be modified by the emulation processor, but only by the host via the modify memory and load commands.

ram - designates memory which can be read or written without restriction.

user - designates memory to be found in the user system.

emulation - designates memory to be supplied by the emulation system.

guarded - designates an address space which is not expected to be accessed. A memory cycle to this space will always attempt to break the processor.

As indicated above, emulation memory may be overlayed for purposes of memory management. Physical memory locations may be given more than one logical address. For example, the following entries are made during memory configuration:

2000H thru 3FFFH emulation ram

0E000H thru 0FFFFH emulation rom overlay 2000H

The screen will show:

Entry	Range	Type	Blocks
1	2000H - 3FFFH	RAM/EMUL	000 - 007
2	E000H - FFFFH	ROM/EMUL	000 - 007

The effect is that emulation memory blocks 000 thru 007 can be accessed either as 2000H thru 3FFFH or as 0E000H thru 0FFFFH. Locations 2000H and 0E000H refer to the same physical location. The designation of ROM or RAM is not significant other than to show the flexibility of this technique.

Table entries may be removed by entering:

delete  $\left\{ \begin{array}{l} \text{<Entry #>} \\ \text{all} \end{array} \right\}$

The memory default is usually guarded, but it may be changed by entering:

default  $\left\{ \begin{array}{l} \text{user} \left\{ \begin{array}{l} \text{rom} \\ \text{ram} \end{array} \right\} \\ \text{guarded} \end{array} \right\}$

A hard copy of the memory map can be obtained at any time during the mapping session by pressing the **[print]** softkey. If no printer is connected to the system, an error message will be displayed.

### Ending the Mapping Session

The memory map configuration session is exited by pressing the **[end]** softkey followed by **[RETURN]**. If an attempt is made to end the mapping session with a blank memory map, an error message will be displayed.

## Simulated I/O Configuration

Simulated I/O configuration begins upon completion of memory configuration. Available host memory for simulated I/O is determined by the number of measurement system modules present. If the maximum number of measurement system modules (4) is present, then simulated I/O memory is not available and the simulated I/O configuration is not presented. If three or less modules are present, then the host memory available is as follows:

one measurement system module, available memory is 768 words.

two measurement system modules, available memory is 512 words.

three measurement system modules, available memory is 256 words.

Available memory is allocated during the actual emulation when an open command is requested for simulated I/O devices. Some devices do not require additional memory. The simulated I/O devices that require memory are: display, printer, RS-232, and disc files.

Each device, except RS-232, requires a minimum of 145 words of memory space. RS-232 requires 128 words of memory space for the read buffer, and 128 words of memory space for the write buffer.

A maximum of five devices, not including RS-232, may be open at one time for a single module measurement system or 768 words available. With RS-232 read and write buffer operation, another three devices may be opened.

A maximum of three devices, not including RS-232, may be open at one time for a dual module measurement system or 512 words available. With RS-232 read and write buffer operations, only one other device may be opened.

A maximum of one device, not including RS-232, may be open at one time for a triple module measurement system or 256 words available. With RS-232 operation, only one read buffer and one write buffer may be open, but no other devices may be opened.

Available memory is deallocated during actual emulation when a close command is requested for the simulated I/O device. Deallocated memory can then be allocated to some other simulated I/O device.

If simulated I/O devices try to allocate more memory than is available, an error return of 9 (request not allowed) is returned to the simulated I/O device control address.

When there is available memory for simulated I/O, the command line displays the following question and answer:

Modify simulated I/O?    no    (yes)

The status line shows:

STATUS: Simulated I/O assignment

Answering "yes" to "modify simulated I/O?" will allow modification to all available simulated I/O devices. The simulated I/O devices are: display, printer, RS-232, keyboard, and up to six disc files.

Questions for a control address for each device are then asked. If a reply of blank is made, then that device is not used. The control address may be specified for a maximum width of 32 bits. The 16 most significant bits, however, must be entered as zeros.

As each question is answered the results are displayed.

The simulated I/O questions are:

- a. display control address?
- b. printer control address?
- c. RS-232 control address?
- d. keyboard control address?

Each unit is identified with a physical address.

Next the command line displays:

modify simulated disc files?    no    (yes)

Answering “no” bypasses any modification to simulated disc files I/O. Answering “yes” allows modification to simulated disc files.

The disc file simulated I/O questions are:

file 1 name?  
file 1 control address?

file 2 name?  
file 2 control address?

file 3 name?  
file 3 control address?

file 4 name?  
file 4 control address?

file 5 name?  
file 5 control address?

file 6 name?  
file 6 control address?

A blank file name disables simulated I/O for the specified file number. Refer to Chapter 8 for further details on simulated I/O.

## **Interactive Measurement Configuration**

It is possible to coordinate measurements between the modules of a multiple module system by selecting various options possible under this category. Since all of these options pertain to the capabilities of the internal analysis card, and are used in conjunction with the trace command, a detailed explanation of these options is included in Chapter 7, along with the other information about internal analysis. Options selected for interaction will be displayed by the measurement system monitor in multiple module systems.

The following question is presented, allowing the user to modify or leave the current interactions unchanged.

Modify interactive measurement specification?    no    (yes)

Allows modification of the internal analysis external inputs and outputs.

If the interactive measurement specification is modified, any function or measurement involving the analysis card will be discontinued. The remainder of the system, however, will not be affected. Any conflict between the interaction specified by a command file and the interaction specified by the measurement system monitor that cannot be resolved, will require modification of the interactive measurement specification for resolution of the conflict.

## Command File Designation

Command file name?     <FILE\_NAME>

This question allows the user to establish a command file containing all of the information pertaining to the questions just answered for emulation configuration. The command file is stored on disc and can then be called up for use during any future emulation session.

All that is required to create the command file is to type in a file name. If no file name is entered, the configuration information will not be stored, and the questions will be required to be answered for each emulation session.

Configuration questions and answers will be stored in a command file of the name specified. Default is the current command file. If no command file exists, a new file will be created under the name provided. Specifying a command file avoids having to answer the configuration questions each time an emulation session is begun. There must be a command file specified for each module in a multi-module emulation session.

Emulation can be started with the same configuration by specifying the `emul_com` file name along with the “emulate” command. The answers to the questions may be changed by specifying “options” “edit” with the “emulate” command. When emulation is ended using the “end” command, the current state of the processor is stored in the `emul_com` file. An additional file of type “trace” is created containing the current analysis specification. This information allows emulation to be re-entered without resetting the processor and analysis hardware. This is done by specifying “options” “continue” in addition to the `emul_com` file name with the “emulate” command. When entering an emulation session through “measurement\_system” and “em6800\_S”, an emulation command file is the only available option. An emulation session within measurement\_system will always be continued, if possible. Editing of an `emul_com` file will be allowed only if there is a conflict, between the configuration file and the hardware, that must be resolved before entering the emulation session.



## Chapter 5

# Operational Commands and System Command Files

## Introduction

Operational commands and system command files are described in this chapter, display/list commands are described in Chapter 6, and analysis commands are described in Chapter 7.

### Command Line Comment Delimiter

The comment delimiter is a semicolon, and is interpreted in such a way that any text following the semicolon, to the end of the command line, will be ignored by the emulation system.

In the example:

```
run from START; causes program execution to begin
```

only the command line text, "run from START", will be acted upon.

## Operational Command Syntax

The syntax listings on the following pages are intended to acquaint the user with the different operational commands. The syntactical variables used in this discussion are described in detail in Appendix A.

# break

## SYNTAX

break

## DEFAULT VALUE

none

### Example:

break

## FUNCTION

“Break” causes the processor to be diverted from execution of the user program to background memory. See Chapter 2 for details of the break function.

# end

## SYNTAX

end
-----

## DEFAULT VALUE

none
------

## Example:

end

## FUNCTION

The “end” command terminates the current emulation session and returns the 64000 operating system to the station monitor mode. The current states of the processor and trace are recorded in the emulation command file and a trace file of the same name. Emulation can then be resumed using the “emulate <CMDFILE> options continue” command. If emulation is terminated using the RESET key, emulation cannot be resumed, and the emulation command file is not overwritten. In a multiple module system, the “end” command returns control to the measurement system monitor program.

# execute

## SYNTAX

**execute** [repetitively]

### Examples:

execute  
execute repetitively

## FUNCTION

“Execute” causes a measurement to begin. The **execute** softkey label will be replaced with the **halt** softkey label whenever a measurement is in progress. If emulation is participating in a system measurement, through cross-triggered analysis or the emulation start function “specify run”, then the global measurement is initiated. Otherwise, a local measurement is begun and “execute” functions identically to “trace again”, i.e., it executes a trace using the previous specification. A measurement can be executed repeatedly by issuing the “execute repetitively” command. This will restart the current measurement after each completion, until the user issues a “halt” command.

A key feature of the “execute” command is that it will start all the modules participating in a system measurement when issued from any one of the modules. If an emulator is started as part of a measurement it will continue running and will not be started again by subsequent executions unless a “specify run” command is again issued. The **execute** softkey is displayed only with multiple module systems.

## SYNTAX

halt
------

### Example:

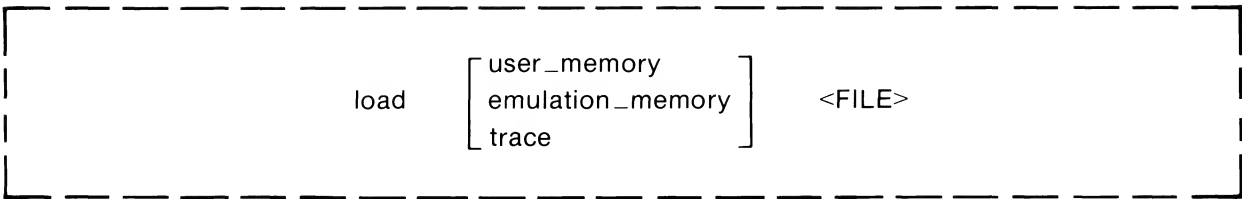
halt

## FUNCTION

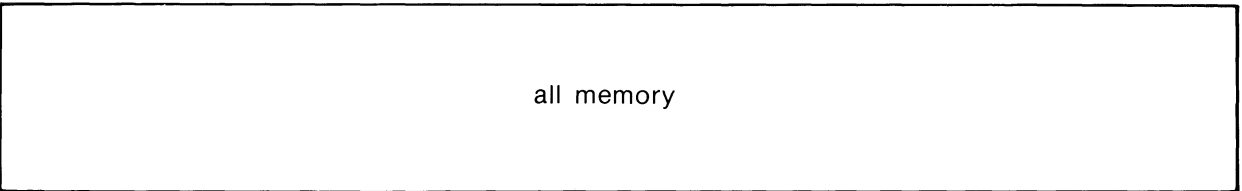
“Halt” causes the measurement currently executing to stop and turns off the “repetitive” option. The **halt** softkey is only displayed during execution in the place of the **execute** softkey. When the “halt” command is performed, some or all of the modules involved may have completed their measurement. “Halt” affects measurements caused by both “trace” and “execute” commands. If emulation is entered with a measurement in progress, “halt” will stop that measurement even if emulation is not interacting in the measurement. The **halt** softkey is displayed only for multiple module systems.

# load

## SYNTAX



## DEFAULT VALUE



## Examples:

```
load KW3000
load emulation_memory KW3000
load trace K5
```

## FUNCTION

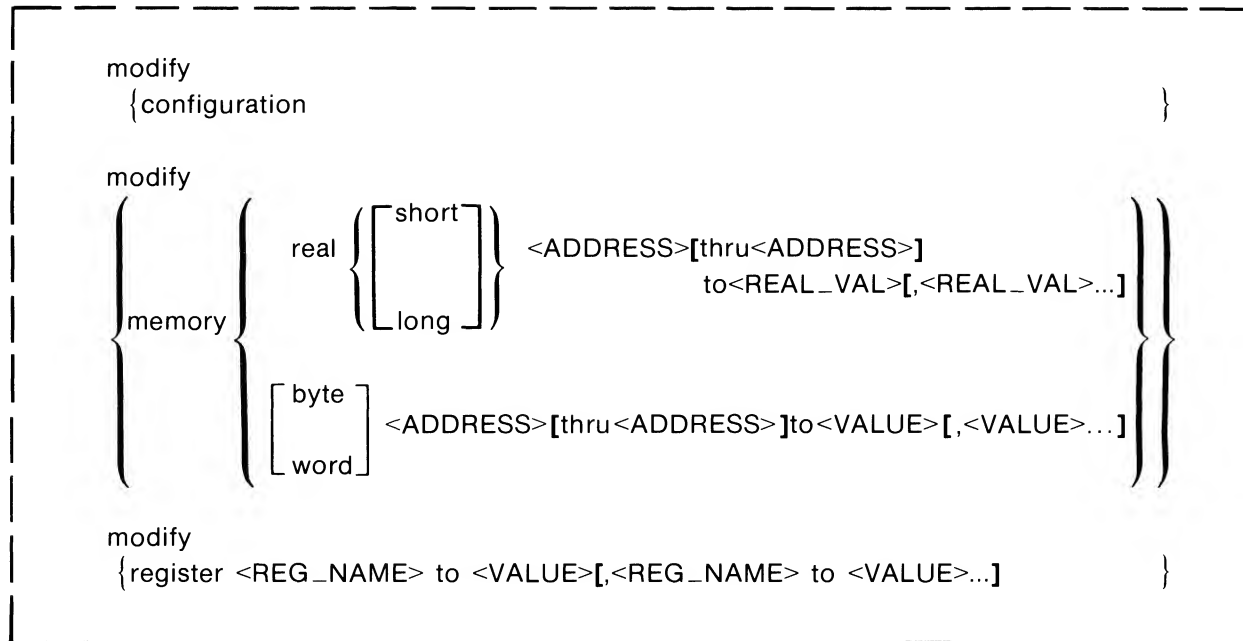
The “load” command transfers absolute code from the 64000 system disc into user RAM or emulation memory. The destination of the absolute code is determined by the memory configuration map which was set up during emulation configuration and the address specified during linking. “Load trace” allows the display command to access and display a previously stored trace. “Load trace” also allows execution of the trace specification via the “trace again” or “execute” commands.

## PARAMETERS

<p>&lt;FILE&gt;</p>	<p>&lt;FILE&gt; is the identifier of the absolute file to be loaded from the 64000 system memory into user RAM or emulation memory or the trace file containing a previously stored trace specification. The syntax requirements for &lt;FILE&gt; are discussed in Appendix A.</p>
---------------------	--

# modify

## SYNTAX



## DEFAULT VALUES

memory real [short]:	if display real is in effect, default is to mode of display, otherwise
[long]	default is to the last mode specified, or to short.
memory [byte]:	if memory display is in effect, default is to mode of display.
[word]	Otherwise, default will be the last value specified, or to byte.

# modify

(Cont'd)

## Examples:

modify configuration

modify memory word 0001H to 8642H

modify memory word 00A0H to 1234H

modify memory byte DATA1 to 0E3H,01H,08H

modify memory DATA1 thru DATA100 to 0FFFFH

modify memory byte ARRAY thru ARRAY+16 to 0,0FFH

modify memory real 0675H to -1.303

modify memory real long TEMP to 0.5532E-8

modify memory real short FIRSTREAL thru LASTREAL to  
1.11E1,2.22E-3,-4.56,9.99E17

modify register A to 39H

modify register B to 0AH, PC to 18H

modify register SP to 13A0H

## FUNCTION

The “modify” command is used to review or edit the configuration, to modify the contents of memory (as integers or as real numbers), or to modify the contents of the processor registers.

# modify configuration

## SYNTAX

```
modify { configuration }
```

## DEFAULT VALUE

none

### Example:

```
modify configuration
```

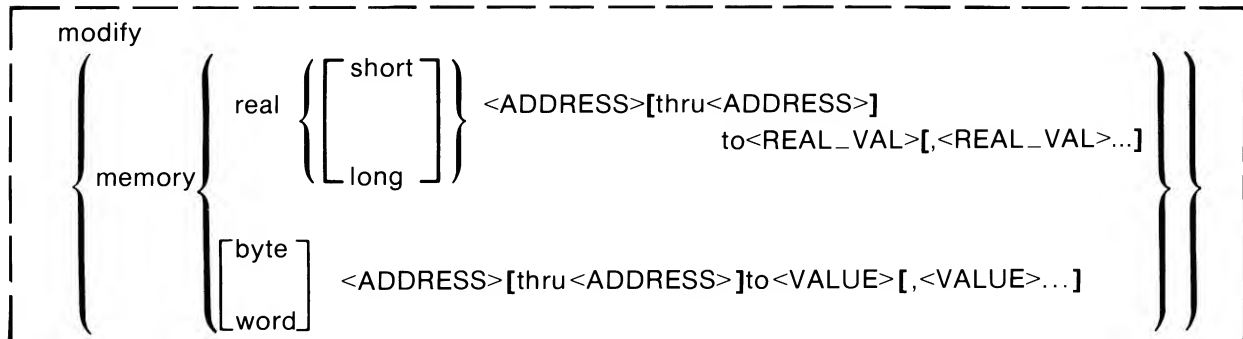
## FUNCTION

The modify configuration command allows the current command file to be reviewed and edited. Each of the configuration questions is presented with the response previously entered. The prior response can be entered as displayed by pressing `RETURN`, or modified as necessary and then entered by pressing `RETURN`.

The command is invoked through the `modify` `config` softkeys.

# modify memory

## SYNTAX



## DEFAULT VALUES

For integer memory modifications, default, initially, is to the “display memory” mode if in effect, otherwise default is to byte; thereafter default is to the “display memory” mode, or else to the last “modify” mode.

For real memory modifications, default is to the “display memory” mode if in effect, otherwise to short; thereafter default is to the “display memory real” mode if in effect, or to the last mode.

## Examples:

```
modify memory word 0001H to 8642H
modify memory word 00A0H to 1234H
modify memory byte DATA1 to 0E3H,01H,08H
modify memory DATA1 thru DATA100 to 0FFFFH
modify memory byte ARRAY thru ARRAY+16 to 0,0FFH
modify memory real 0675H to -1.303
modify memory real long TEMP to 0.5532E-8
modify memory real short FIRSTREAL thru LASTREAL to
1.11E1,2.22E-3,-4.56,9.99E17
```

## FUNCTION

The “modify memory” command can modify the contents of each memory location in a series to an individual value or the contents of all of the locations in a memory block to single or repeated sequence of values.

# modify memory

(Cont'd)

## PARAMETERS

<ADDRESS>	<ADDRESS> determines which memory location or series of locations are to be modified.
<VALUE>	<VALUE> is the number which is to be loaded into the specified memory location or locations. The syntax for <VALUE> is described in Appendix A.
<REAL_VAL>	<REAL_VAL> is the real number value to be loaded into the specified memory location or locations. The syntax for <REAL_VAL> is described in Appendix A.

## DESCRIPTION

A series of memory locations is modified by specifying the address of the first location in the series to be modified (<ADDRESS>) and the list of the <VALUE>s, including <REAL\_VAL>s, to which the contents of that location and the succeeding locations are to be changed. Both bytes must be addressed if a memory word is to be modified. The first <VALUE> listed replaces the contents of the specified memory location, the second <VALUE> replaces the contents of the next location in the series, and so on until the list has been exhausted. If only one number or symbol is specified, only the single address indicated is modified. When more than one <VALUE> is listed, the <VALUE> representations must be separated by commas.

An entire block of memory can be modified such that the contents of each location in the block is changed to the single specified <VALUE>, or to a single or repeated sequence. This type of memory modification is achieved by entering the limits of the memory block to be modified (<ADDRESS> thru <ADDRESS>) and the <VALUE> or list of values, <VALUE>,...,<VALUE>, to which the contents of all locations in the block are to be changed.

# modify register

## SYNTAX

```
modify
    {register <REG_NAME> to <VALUE>[,<REG_NAME> to <VALUE>...]}
```

## DEFAULT VALUE

none

## Examples:

```
modify register A to 39H
modify register B to 0AH, PC to 18H
modify register SP to 13A0H
```

## FUNCTION

The “modify register” command is used to modify the contents of one or more of the microprocessor’s internal registers. The entry for <REG\_NAME> determines which register is modified.

Register modification cannot be performed during real-time running of the processor. A break must be performed to gain access to the processor registers.

## PARAMETERS

<VALUE>	<VALUE> is the number which is to be loaded into the specified processor register. The syntax for <VALUE> is described in Appendix A.
<REG_NAME>	<REG_NAME> represents the name of one of the registers to be modified. The possible entries for <REG_NAME> are shown in the heading on the register display.

# reset

## SYNTAX

reset
-------

## DEFAULT VALUE

none
------

## Example:

reset

## FUNCTION

“Reset” suspends target system operation and reestablishes initial operating parameters, such as reloading control registers.

# run

## SYNTAX

```
run [from <ADDRESS>] [until <UNTIL_TRIGGER>]
```

## DEFAULT VALUE

<b>&lt;ADDRESS&gt;</b>	<b>&lt;ADDRESS&gt;</b> option may be an address or a label. If the <b>&lt;ADDRESS&gt;</b> option is omitted, the emulator will begin program execution at the current address specified by the processor's program counter, or, if an absolute file containing a transfer address has just been loaded, execution will start at that address.
------------------------	---

Where **<UNTIL\_TRIGGER>** is defined as:

**<STATE>** [occurs <# times>] [or **<STATE>**]  
**<RANGE\_STATE>** [occurs <# times>]

See the "trace" command syntax for definitions of **<STATE>** and **<RANGE\_STATE>**.

## Examples:

```
run
run from 1000H
run from COLD_START
run until 0AFFH
run until 1FFH thru 20FH occurs 3 times
```

## FUNCTION

If the processor is in a reset or break state, "run" will cause the processor to begin executing from the Next PC, and if a "from" address is specified the processor will be directed to that address. The program can either be run from a specified **<ADDRESS>** or from the address currently stored in the processor's program counter, or from a label specified in the program.

**PARAMETERS**

- from <ADDRESS>      from <ADDRESS> represents a state on the address bus which can be used to start a program run. The syntax requirements for <ADDRESS> are equivalent to those for <VALUE> as defined in Appendix A.
- until <UNTIL\_TRIGGER>      uses internal analysis to cause an exit from a user program to background memory when a state satisfying the <UNTIL\_TRIGGER> term is encountered.

# specify

## SYNTAX

```
specify {run [from <ADDRESS>] }  
        { <TRACE_COMMAND> }
```

### Examples:

```
specify run from START  
specify trace after address 1234H
```

## FUNCTION

“Specify” is used to prepare a “run” or “trace” command for execution, and is used in conjunction with the “execute” command. If the processor is not reset, then “specify run” causes a break from a user program, and initializes the PC to the default address or to the specified address. An “execute” command will then cause the run to occur. Once an execution has occurred, the run specification is removed and can not be repeated without respecifying the run.

If the processor is reset and no address is specified, then an “execute” will cause the processor to run from the next condition. If the processor is reset from specified address, then the processor is allowed to run and the next program count is set up for the specified address.

“Specify trace” causes the trace hardware to be initialized with the given trace specification. An “execute” command will then cause the trace to be executed. A trace specification is not removed and can be reexecuted without another “specify trace” command. “Specify trace” and “specify run” can be used with a single “execute” command initiating both the run and the trace, but this mode can only be used if the internal analysis is configured to participate in a system measurement. If internal analysis is not configured, then “specify trace” and “specify run” are mutually exclusive and issuing one after the other will negate the first command. If “specify trace” is followed by “execute”, the effect is identical to “trace”. If “specify run” is followed by “execute”, the effect is the same as “run”, except that if a system measurement is configured, it is initiated. The specify softkey label is displayed only with multiple module systems.

## SYNTAX

```
step [<# STEPS>][from <ADDRESS>]
```

## DEFAULT VALUES

<# STEPS>	If no value is entered for number of steps, only one instruction is executed each time the <code>RETURN</code> key is pressed. Multiple instructions can also be executed by holding down the <code>RETURN</code> key.
from <ADDRESS>	If the from <ADDRESS> option is omitted, stepping begins at the next program counter address.

## Examples:

```
step
step from 1000H
step 20 from 2000H
```

## FUNCTION

The “step” command allows program instructions to be sequentially analyzed by causing the emulation processor to execute a specified number of instructions. The contents of the processor registers and the contents of emulation or user memory can be displayed after each “step” command has been completed.

## PARAMETERS

<# STEPS>	<# STEPS> determines how many instructions will be executed by the step command. The number of instructions to be executed can be entered in binary (B), decimal (D), octal (O or Q), or hexadecimal (H) notation.
from <ADDRESS>	from <ADDRESS> represents a state on the address bus which can be used to start a program run. The syntax requirements for <ADDRESS> are equivalent to those for <VALUE> as defined in Appendix A.

# stop\_trace

## SYNTAX

stop_trace
------------

## DEFAULT VALUE

none
------

### Example:

stop\_trace

## FUNCTION

The “stop\_trace” command terminates the current trace, and stops the execution of the current measurement. That is, the system stops searching for trigger and trace states. Trace memory, although incomplete, can be displayed. “Stop\_trace” will also halt internal analysis if it is being used in “run until” mode.

The command is invoked through the stop\_trc softkey.

# store

## SYNTAX

```
store { memory <ADDRESS> thru <ADDRESS> } to <FILE>
      trace
```

## DEFAULT VALUE

none

### Examples:

```
store 800H thru 20FFH to TEMP2
store EXEC thru DONE to TEMP3
store trace to TRACE
```

## FUNCTION

The “store” command is used to store the contents of specific memory locations in an absolute file, or the trace memory in a trace file.

## PARAMETERS

<ADDRESS>	<ADDRESS> determines the memory locations from which data is to be stored into the specified absolute file.
<FILE>	<FILE> is the identifier for the absolute file or trace file in which data is to be stored. The syntax requirements for <FILE> are described in Appendix A.

## DESCRIPTION

<FILE> determines the name under which the absolute or trace file is to be stored. The “store” command creates a new file having the specified name as long as there is no absolute file with that name presently on the disc. In the cases where a file represented by the <FILE> variable already exists, the system asks whether the old file is to be deleted. If the response is “yes”, the new file replaces the old one. If the response is “no”, then the “store” command is cancelled and no data is stored. Transfer address of absolute file is set to zero.

# System Command Files

System command files can be used within an emulation session, but must be constructed before the emulation session begins. A softkey prompt allows insertion of the system command file into program execution.

A system command file can be constructed by using the following procedure:

- a. From the system monitor level, issue the command “log\_commands to NEW”.
- b. Enter emulation session.
- c. Proceed thru all desired commands.
- d. End emulation, return to the system monitor level and “log\_commands off”.
- e. Edit NEW (the command file just created) by deleting the undesired commands or making any changes needed.
- f. The system command file is now ready for use within the emulation session.

## <CMDFILE>

### SYNTAX

<CMDFILE> [PARMS]

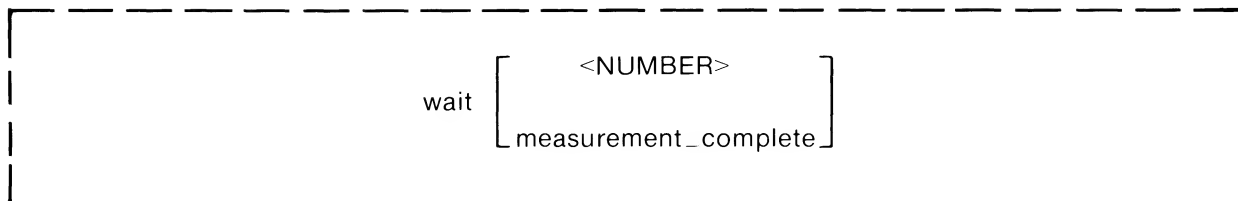
### FUNCTION

<CMDFILE> is the system command file name and is further described in Appendix A. The use of [PARMS] is described in the system manual under command files.

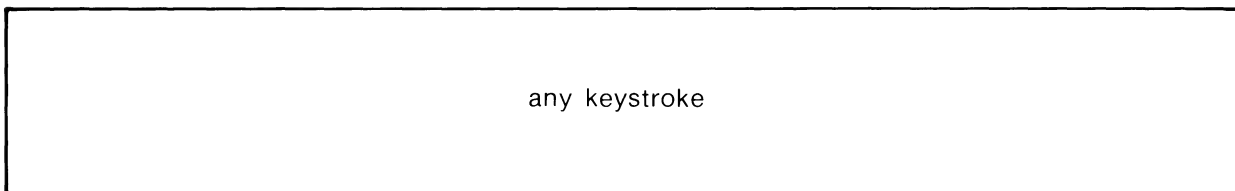
# Command Delays

## wait

### SYNTAX



### DEFAULT VALUE



### Examples:

wait                      will wait for any keystroke before accepting the next command.

wait 6                    will wait for any keystroke or 6 seconds before accepting the next command.

wait measurement\_complete                      will wait for any keystroke or for a pending measurement to become complete. If no measurement is in progress, wait will be satisfied immediately.

### FUNCTION

Command delays are enhancements that allow flexible use of system command files.

# wait

(Cont'd)

## PARAMETERS

<NUMBER> is a number of seconds (65,535 maximum) before the next command is executed.

measurement\_complete is a delay until a measurement has been completed before the next command is executed.

When operating in REMOTE mode, "wait" for keystroke only is not allowed. A <NUMBER> or measurement\_complete term must be included with the "wait" command. Pressing the system RESET key will satisfy the "wait" for keystroke condition and will stop execution of a command file, if a command file is currently being executed.

## DESCRIPTION

The usefulness of command delays lies in the capability to give the emulation system and target processor time to reach some condition or state before bringing in the next command. The delay commands may be included in the system command file.

The following example shows the use of wait commands within a system command file.

```
load PROGRAM
run from SUB1
trace about BEGINNING
wait measurement_complete
trace only address range DATASTART thru DATAEND
run from SUB2
wait 8
stop_trace
list FILE1 trace
run from SUB3
```

Run from subroutine 1 and accept the next command after measurement is completed. Trace in DATA area while running subroutine 2, then list to a file after subroutine 2 has been completed. Wait 8 allows the processor 8 seconds before the stop\_trace becomes effective.

## Display and List Commands

### Display and List Command Capabilities

There are four basic types of information which may be viewed by using either the “display” or “list” command. These are:

Memory data

Register contents

Trace information

Global and local symbols

#### Memory Data

For data taken from memory, the starting address in memory or a list of memory address ranges may be specified.

Whether the data comes from emulation or user memory depends upon the memory map assignments made during configuration of the emulation command file. Unless otherwise specified, memory data is displayed statically with the actual memory address shown. (The static display shows the memory contents existing when the display command is executed.) The data is displayed in hexadecimal form with corresponding ASCII characters as shown in Figure 6-1.



The display address will increment or decrement by units of one when using the up arrow or down arrow keys to view memory data in the mnemonic format. In this way the currently displayed mnemonic page can be aligned via inverse assembly, beginning at a new starting address. The roll up (or roll down) key in a mnemonic display will disassemble the next (or previous) address from the last (or first) displayed address, leaving the rest of the display unchanged. (Roll up and up arrow, and roll down and down arrow keys, are equivalent in either absolute or blocked modes.)

The `next page` and `prev page` keys will replace all of the data with new data. The `next page` will place the next instruction address and succeeding instruction addresses and corresponding data on the screen. The `prev page` key will place the preceeding instruction addresses and corresponding data on the screen. In some cases, in the "prev page" mode, there may be a slight delay before the data is placed on the screen. The delay results when the system steps backwards through the memory until sufficient data has been gathered to fill the screen.

```

Memory      :mnemonic
address
1072  DRAB   0413H,E
1075  BEQ    101FH
1077  LDAA   0407H,E
107A  CMPA   #00H
107C  BEQ    1081H
107E  JMP    108DH,E
1081  LDX    #0407H
1084  JSR    10ACH,E
1087  LDX    #0406H
108A  JSR    12D3H,E
108D  JSR    1571H,E
1090  STX    10AAH,E
1093  LDX    0402H,E
1096  STX    1233H,E
1099  LDAB   #80H
109B  LDX    1233H,E

STATUS: 6800/2--Reset in background                      11:50

run trace step display modify break end ---ETC---
```

Figure 6-2. Memory Contents - Mnemonic

- c. Real number display/list. Data may be viewed as real numbers in either the short form (four bytes) or the long form (eight bytes).
- d. Memory addresses may be displayed "offset" from the actual value. The address offset allows the actual addresses to be offset by a value specified by the user. If the value is correctly chosen, the address space displayed will start at location 0000H and will correspond to the listing generated by the assembler or compiler. For example, if a module originating at address X is linked with other modules, it may be assigned a new starting address X+Y where Y is a value that depends on the number and size of the other modules being linked. Offset, therefore, allows the user to enter "Y" so that the addresses appear the same as in the assembly or compiler listing file.

## Register Contents

Register data is displayed as shown in Figure 6-3. The program counter (PC) value can be offset by a specified value and the next program counter (Next\_PC) value will be offset by an equal amount. The offset is done for the same reason as described above for memory data.

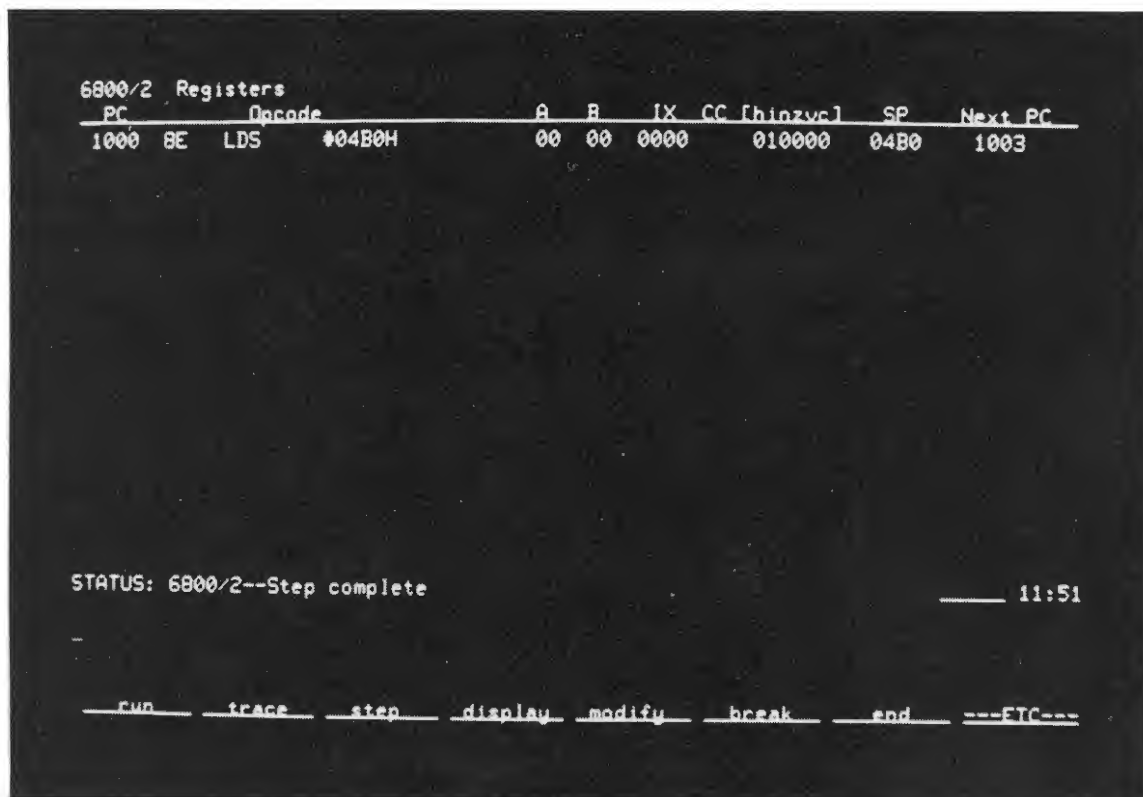


Figure 6-3. Register Contents

## Trace Information

Trace information may also be displayed or listed using the display/list command. Figure 6-4 shows a trace memory display.

Trace:		mnemonic		break: none		count:	
line#	address	opc	data	mnemonic	opcode	or status	time, relative
-007	1016	04				operand or data read	1. uS
-006	1017	07				operand or data read	1. uS
-005	0407	00				operand or data read	1. uS
-004	1018	C1		CMPB	#00H		1. uS
-003	1019	00				operand or data read	1. uS
-002	101A	27		BEQ	101FH		1. uS
-001	101B	03				operand or data read	1. uS
6800	101F	BD		JSR	134CH,E		3. uS
+001	1020	13				operand or data read	1. uS
+002	1021	4C				operand or data read	1. uS
+003	134C	CE				operand or data read	1. uS
+004	04B0	22				write	1. uS
+005	04AF	10				write	1. uS
+006	1021	4C				operand or data read	3. uS
+007	134C	CE		LDX	#13DFH		1. uS
+008	134D	13				operand or data read	1. uS

STATUS: 6800/2--Running                      Trace complete                      11:55

\_\_\_\_ cmd \_\_\_\_ trace \_\_\_\_ step \_\_\_\_ display \_\_\_\_ modify \_\_\_\_ break \_\_\_\_ end \_\_\_\_ ---ETC---

Figure 6-4. Trace Memory Display

## Global and Local Symbols

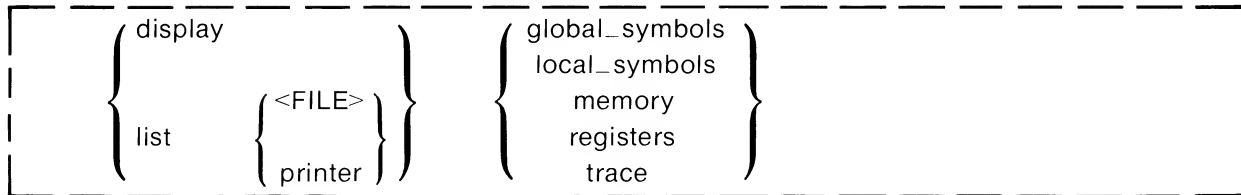
These symbols may be viewed on the display. Local symbols are symbols defined in the source file for a single program module. Global symbols are those that are declared to be global in any source file. They are defined using the assembler pseudo instruction, GLB (or \$GLOBVAR+\$ in the compiler). When the display command is used to examine either of these symbol types, the display will contain the symbol name, absolute address, and, for symbols located in emulation memory, their present value (and for local symbols the relative value of PROG, DATA, COMN). If the processor is running and is restricted to real-time runs, the values are displayed as asterisks (\*\*).

## **Display and List Command Syntax**

The “display” and “list” commands initiate the display of local or global symbols, the contents of registers or memory, the contents of the trace memory. For the purpose of this discussion, “display” and “list” command options are treated as separate commands and are described as such on the following pages.

# display/list

## SYNTAX



## DEFAULT VALUE

Depending on what is listed, defaults may be the options selected for the previous execution of the “list” or “display” command.

## Examples:

```
list printer memory 001FH thru 005FH
display registers
list printer trace
list JIM local_symbols_in KEEP:USER
list printer global_symbols
list printer memory --- (defaults to current information on the display.)
```

## FUNCTION

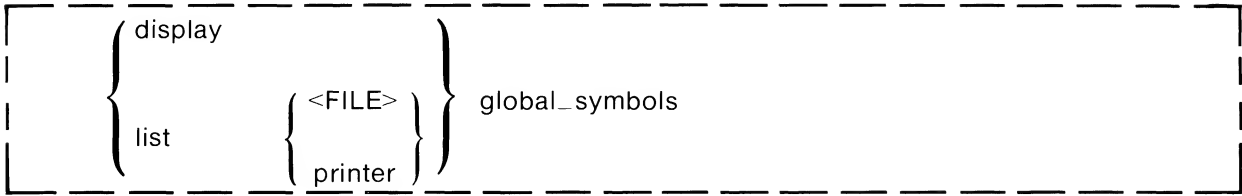
The “list” command produces a copy of the information selected. The “display” command displays the information and allows the use of the `ROLL UP` , `ROLL DOWN` , `PREV PAGE` , `NEXT PAGE` , and in some cases the up arrow and down arrow keys. The copy resulting from a “list” command can be either a listing file stored in the 64000 memory or a hard copy produced by the printer. If the information is written to an existing file, the old file is overwritten by the new information.

## PARAMETERS

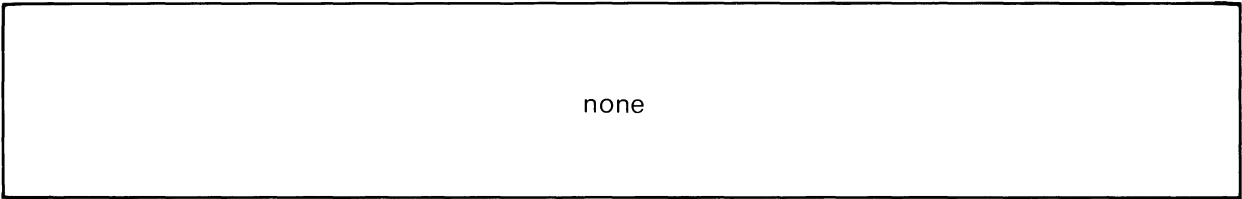
printer	printer causes a hard copy to be printed.
<FILE>	<FILE> causes the information to be copied to either a new or an existing file identified by <FILE>. The syntax for <FILE> is discussed in Appendix A.

# display/list global\_symbols

## SYNTAX



## DEFAULT VALUE



## Examples:

display global\_symbols

list JOE global\_symbols

## FUNCTION

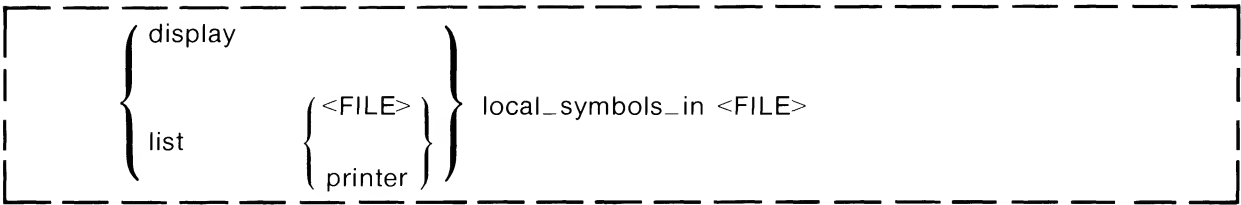
The “display/list global\_symbols” command displays the global symbols defined for the current absolute file and the logical addresses and present values of those symbols. Global symbols are looked up in the link\_sym file which is generated during linking. If the link\_sym file is not present, no symbols may be displayed or used in expressions. Global symbols are those that are declared to be global in the source file. When the “display/list global\_symbols” command is used, the listing will include the symbol name, address, and its present value. The present values of symbols in emulation memory will be displayed. An asterisk (\*) will be displayed in the value field for other symbols.

## PARAMETER

glb\_symb                      glb\_symb represents the symbols and labels defined as global in one of the source programs from which the current absolute file was generated. When the glb\_symb softkey is pressed, “global\_symbols” is displayed on the screen.

# display/list loc\_symb

## SYNTAX



## DEFAULT VALUE

none
------

## Examples:

```

display local_symbols_in TEMP1
list printer local_symbols_in TEMP1
list BOB local_symbols_in TEMP1

```

## FUNCTION

The “display/list loc\_symb” command displays the local symbols and their present values and relative mode as defined in the source (program, data, or common) <FILE>. Local symbols are looked up in the asmb\_sym file generated during assembly or compilation. If the asmb\_sym file is not present, no local symbols may be displayed or used in expressions.

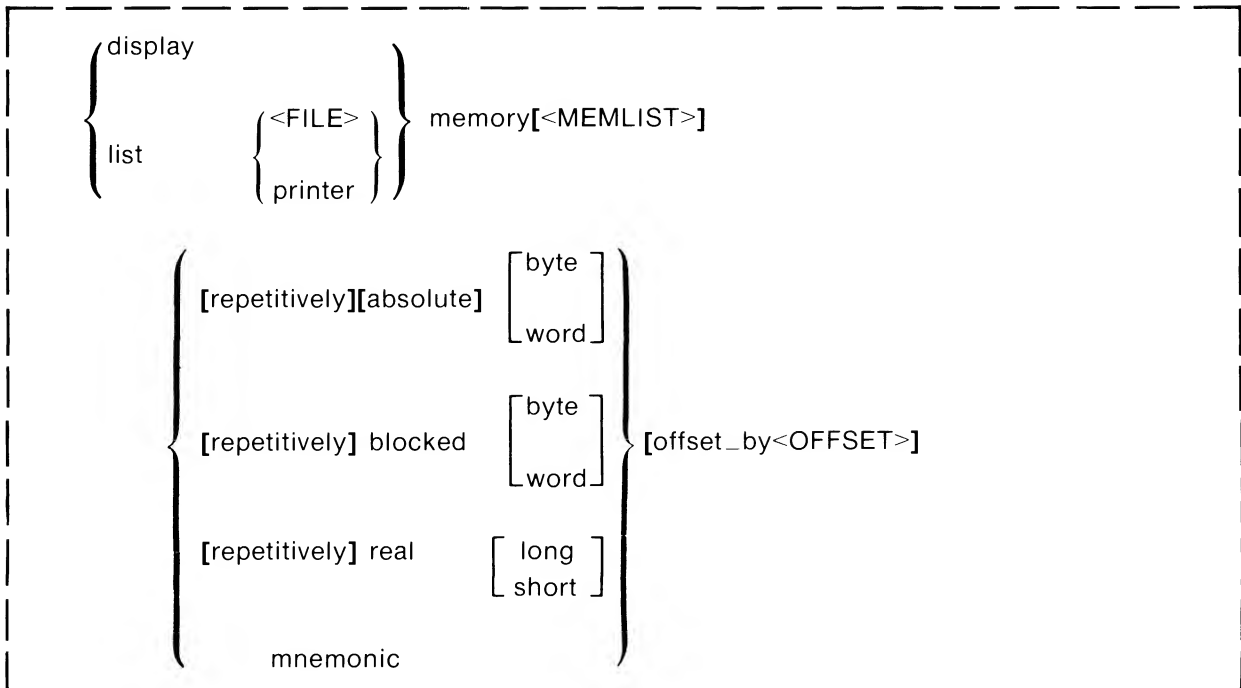
The present values of symbols in emulation memory will be displayed. An asterisk (\*) will be displayed in the value field for other symbols.

## PARAMETERS

loc_symb	loc_symb refers to the symbols and labels defined as local in the source file identified by <FILE>. When the <span style="border: 1px solid black; padding: 0 2px;">loc_symb</span> softkey is pressed, “local_symbols” is displayed.
<FILE>	<FILE> represents the source file that contains the local symbols to be displayed. Refer to Appendix A for the syntax requirements of <FILE>.

# display/list memory

## SYNTAX



where <MEMLIST> is defined as:

`<ADDRESS> [thru <ADDRESS>] [, <ADDRESS> [thru <ADDRESS>] ...]`

## DEFAULT VALUES

Initial values are the same as specified by the command “display memory 0 blocked byte offset\_by 0”.

Defaults are to values specified in the previous “display memory” or “list memory” command.

“Repetitively” must be specified each time “display memory” is issued.

# display/list memory

(Cont'd)

## Examples:

display memory START mnemonic

display memory 0 thru 100H, START thru START+5,  
500H, TARGET1, TARGET2 blocked word

list memory 810H offset\_by @:MODULE1

## FUNCTION

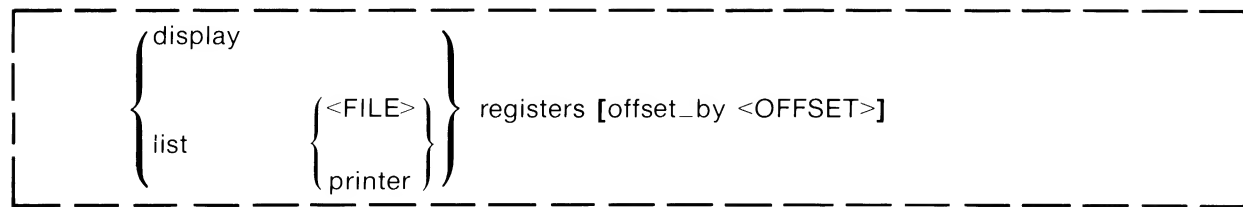
The “display/list memory” command shows the contents of the specified memory location or series of locations. The memory contents can be viewed either statically or repetitively (display memory only) and either in mnemonic or hexadecimal form. In addition, the memory addresses can be displayed offset by a value which allows the information to be easily compared to the file listing.

## PARAMETERS

<MEMLIST>	<MEMLIST> describes the addresses of memory to be displayed. It consists of either a single address, in which case the memory display starts with that address, or a list of single addresses or ranges of addresses.
repetitively (display only)	repetitively causes the display to be periodically updated with the current contents of memory. The program must be interrupted in order to fetch the memory data and update the display (doing so one line at a time).
mnemonic	mnemonic causes the program in memory to be disassembled. The mnemonic opcodes, memory locations, and associated operands are then displayed or listed.
<OFFSET>	<OFFSET> causes the system to subtract the specified <OFFSET> from each of the actual absolute addresses before the addresses and the corresponding memory contents are displayed. The value of <OFFSET> can be selected such that each module in a program appears to start at address 0000H. The “display/list” of the memory contents will then appear similar to the assembly or compiler listing.

# display/list registers

## SYNTAX



## DEFAULT VALUE

<OFFSET>	Initially 0; thereafter previous value.
----------	---

### Examples:

```
display registers
display registers offset_by 810H
list JIM registers offset_by 0A10H
```

## FUNCTION

The “display/list registers” command gives program counter value, the current contents of the processor’s registers, and, if a step has just been executed, the mnemonic of the last instruction. This process does not occur in real-time; therefore, if the registers are to be displayed while the processor is running, the system must be configured to allow nonreal-time operations.

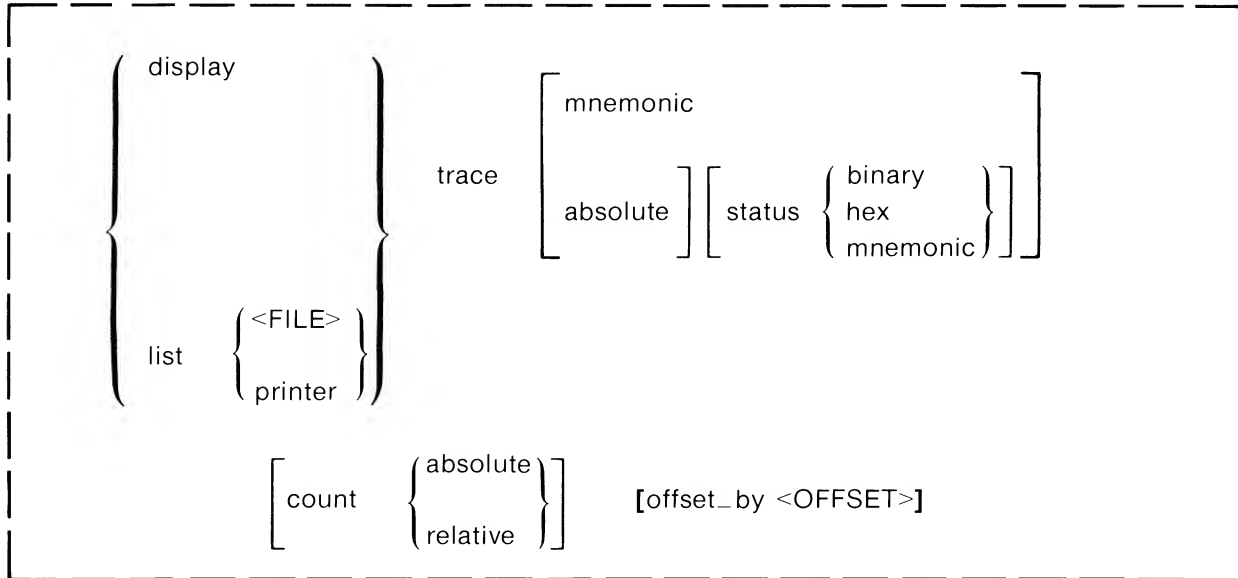
The displayed values of both the program counter and the next program counter can be offset from their actual values by a number that allows the register information to be easily compared to the assembled or compiled listing.

## PARAMETERS

<OFFSET>	<OFFSET> represents the value by which the displayed program counter (PC) and next program counter (Next_PC) addresses are offset from their actual values. The syntax for <OFFSET> is equivalent to the syntax for <VALUE> as described in Appendix A.
----------	---

# display/list trace

## SYNTAX



## DEFAULT VALUES

Initial values are the same as specified by the command “display trace mnemonic count relative offset\_by 0”.

<OFFSET>      Initially 0; thereafter previous value.

## Examples:

```
display trace count relative  
display trace status binary  
list EXEC trace count absolute  
list printer trace offset_by 0100H
```

## FUNCTION

The “display/list trace” command shows the contents of the trace buffer. The information can be presented as absolute hexadecimal code or in mnemonic form. The status captured by the analyzer can be displayed mnemonically, independent of the address and data information, or it can be displayed in hexadecimal or binary form. Addresses captured by analysis are physical addresses.

Refer to Figure 6-4 for an example of a “display trace count relative” command.

# display/list trace

(Cont'd)

The “offset\_by” option causes the system to subtract the specified <OFFSET> from the addresses of the executed instructions before the trace is displayed. With an appropriate entry for <OFFSET>, each instruction in the displayed trace will appear as it does in the assembled or compiled program listing.

The “display/list count” command is used, after a trace has been obtained, to change the current display of time or state counts to one in which the counts are displayed either relative to the previous event or as an absolute count measured from the trigger event. If time counts are currently selected, the “display count” command causes an absolute or relative time count to be displayed. If the current display contains state counts, a relative or absolute state count results.

## PARAMETERS

mnemonic	mnemonic directs the system to display trace information with opcodes in mnemonic format.
absolute	absolute directs the system to display the status information rather than mnemonic opcodes.
status	
hex	displays status information in hexadecimal form.
binary	displays status information in binary form.
mnemonic	displays status information in mnemonic form.
<OFFSET>	<OFFSET> represents the number by which the address displayed for an executed instruction is offset from the instruction's actual address. The syntax for <OFFSET> is equivalent to the syntax for <VALUE> as described in Appendix A.
count	
absolute	absolute causes the state or time count for each event of the trace to be displayed as the total count measured from the trigger event.
relative	relative causes the state or time count for each event of the trace to be displayed as the count measured relative to the previous event.

## Analysis and Interactive Commands

### Introduction

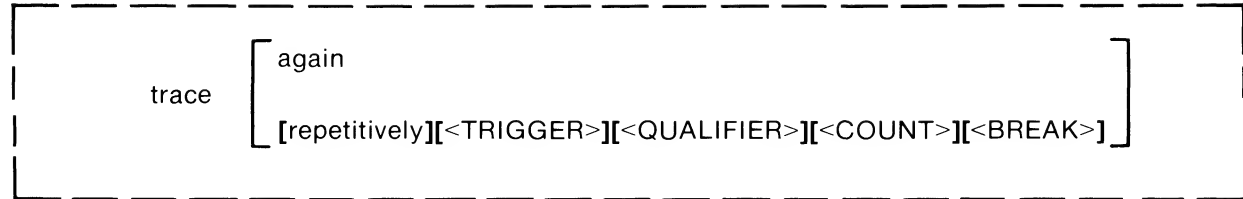
The analysis commands are used to specify the particular part of a program that is to be traced and displayed. The trace measurement may be made once and displayed statically or the same measurement may be made repetitively and the results continually updated.

The “trace” command causes 256 states to be collected and stored in the trace memory. The trace memory is displayed relative to the trigger position. The trigger may occur at the beginning (after), at the middle (about), or at the end (before) of the trace memory contents. Note that the display is capable of listing only 16 lines per page, and therefore the `PREV PAGE`, `NEXT PAGE`, `ROLL UP`, or `ROLL DOWN` keys are used to view all measured states.

Emulation can interact with other modules of a multiple module system over the intermodule bus, or with external equipment through the BNC ports. Commands that involve interaction are: specify, execute, trace, stop\_trace, and halt. Emulation can participate in coordinated measurements and can also begin execution of a program in concert with the initiation of a measurement. Chapter five contains details for “specify”, “execute”, “stop\_trace”, and “halt”. Details of measurement interaction possibilities appear in this chapter under the heading “Interactive Measurement Selection”. Details of the “trace” command follow.

# trace

## SYNTAX



where <TRIGGER> is defined as:

$$\left\{ \begin{array}{l} \text{after} \\ \text{about} \\ \text{before} \end{array} \right\} \left\{ \begin{array}{l} \text{<STATE> [occurs <\#TIMES>] [or <STATE>]} \\ \text{<RANGE\_STATE> [occurs <\#TIMES>]} \end{array} \right\}$$

<QUALIFIER> is defined as:

$$\left\{ \begin{array}{l} \text{<STATE> [or <STATE>]} \\ \text{<RANGE\_STATE>} \end{array} \right\}$$

<COUNT> is defined as:

$$\text{counting} \left\{ \begin{array}{l} \text{state <STATE>} \\ \text{time} \end{array} \right\}$$

<BREAK> is defined as:

$$\text{break\_on} \left\{ \begin{array}{l} \text{measurement\_complete} \\ \text{trigger} \end{array} \right\}$$

<RANGE\\_STATE> is defined as:

$$\text{address} \left\{ \begin{array}{l} \text{range <VALUE> thru <VALUE>} \\ \text{not range <VALUE> thru <VALUE>} \\ \text{not <VALUE>} \end{array} \right\}$$

[ data<VALUE> ] [status<STATUS\_EXPRESSION>]

<STATE> is defined as:

$$\left\{ \begin{array}{l} \text{address } \langle \text{VALUE} \rangle \text{ [ data } \langle \text{VALUE} \rangle \text{ ] [ status } \langle \text{STATUS\_EXPRESSION} \rangle \text{ ]} \\ \text{data } \langle \text{VALUE} \rangle \text{ [ status } \langle \text{STATUS\_EXPRESSION} \rangle \text{ ]} \\ \text{status } \langle \text{STATUS\_EXPRESSION} \rangle \end{array} \right\}$$

<STATUS\_EXPRESSION> is defined as:

$$\left\{ \begin{array}{l} \langle \text{STATUS\_IDENT} \rangle \\ \langle \text{VALUE} \rangle \end{array} \right\} \text{ [and } \langle \text{STATUS\_EXPRESSION} \rangle \text{]}$$

Figure 6-4 shows the result of a trace specification consisting of “trigger” (about), “address” (hexadecimal), “qualifier” (opcode), “count” (time), and no “break”.

A shorthand syntax may be used when entering the information required by the <STATE> variable. The words “address”, “data”, and “status” can be omitted as long as commas are used to separate the fields which contain the entries for each state. For example, “address 810H data 0FFH status 14H” could be entered as follows: “810H,0FFH,14H”. Likewise, “address 810H status 14H” could be entered as “810H,,14H” using the shorthand syntax. Notice that when a particular field has no entry, commas must still be used to separate the fields. The first comma specifies the end of the address field, and the second comma specifies the end of the data field.

The trigger and qualifier parts do not have the entire syntax described above. Only one may have a range on address and only one may have an “or”ed term. The softkeys and grammar reflect this and will not allow entry of illegal specifications.

In all cases the term <VALUE> is an expression consisting of addition, subtraction, multiplication, division, parentheses, numbers, and symbols. In hexadecimal, binary, and octal numbers don’t cares (X) may be used. They may not, however, be combined with arithmetic operations and may not be used in the address <VALUE> of a <RANGE\_STATE>.

## Status “and” Function

<STATUS\_IDENT> is any one of the predefined mnemonic status values. Using “and” capability, status identifiers and/or values can be combined. It is possible, for example, to enter status 00000000B and status 11111111B; a combination that will result in the error message, “Status expression error”.

The “and” function for status expressions operates bitwise on values entered, or on the predefined values of the mnemonic status identifiers. Table 7-1 defines the results of the “and” function for any bit.

**Table 7-1. "And" Function Results**

	X	0	1
X	:	X	0 1
0	:	0	0 E
1	:	1	E 1

Where X is the symbol for a "don't care" bit, and E represents an invalid entry that will result in the message "Status expression error".

## Using Analysis Commands

Analysis may be performed either by first initiating the program run and then specifying the trace parameters or by specifying the trace parameters first and then initiating the program run. In either case, once a trace command is initiated, the analysis module monitors the system buses of the emulation processor to detect the states specified in the trace command. When the trace specification has been satisfied, a message will appear on the status line showing "trace complete". At that time the contents of the trace memory can be displayed. If the trace memory contents exceed the page size of the display, the `NEXT PAGE`, `PREV PAGE`, `ROLL UP`, or `ROLL DOWN` keys may be used to display all the trace memory contents.

Trigger and storage qualification can be specified without initiating a trace by using the "specify trace" command, and traces can be initiated without altering the trigger and storage qualifications by using the "execute" command.

The "trace" command consists of the components described in the following paragraphs:

- a. <TRIGGER> - The "trigger" is the event on the emulation bus to be used as the starting, ending, or centering event for the trace.
- b. <QUALIFIER> - The storage specification determines which of the traced states will be stored in the trace memory for display upon completion of the trace. The trace memory can be filled by those states which occur immediately before or immediately after the specified trigger event, or half of the memory can be filled by states which precede the trigger and half by those which follow the trigger event. Events can be selectively saved by pressing "trace only" and entering the specific events to be saved. When this option is used, only the indicated states occurring in the specified position relative to the trigger are stored in the trace memory.

- c. <COUNT> - The count option specifies whether time or the occurrence of a state will be counted during the trace. The data can be displayed either “relative” to the count at the previous stored state, or “absolute” with respect to the trigger. All count measurements can be displayed in either absolute or relative mode. The absolute count is the total count from the trigger to each measured state. A plus sign (+) preceding the trace number indicates that the state occurred after the trigger state. A minus sign (-) indicates that the state has occurred before the trigger state.

The “relative count” mode displays the count between consecutive states stored in the trace buffer. It can be used to measure execution times of subroutines and instructions or the time between the occurrence of the same state in the execution of a program.

- d. <BREAK> - The break specification causes an exit from the executing program to the background at a predetermined point in the emulation program.
- e. again - Entry of the “again” parameter causes the trace to be performed again using the previous trace parameters.
- f. repetitively - Entry of the “repetitively” parameter causes a new trace to be initiated after the results of the previous trace are displayed. The trace will continue until a “stop\_trace” or a new “trace” command is issued.

## Interactive Measurement Selection

The internal analysis unit can interact with other measurement equipment during emulation through either or both of the BNC output ports located on the back of the development station. The analysis unit can also interact with other cardcage analysis modules through the IMB connector located at the top of the analysis card. The following questions appear during configuration.

Modify interactive measurement specifications?    no    (yes)

If interaction is desired or if a previously defined interactive specification is to be modified, this question allows the analysis interaction specification format to be reviewed and modified as necessary. If no modification is desired, the “no” response should be selected. The Interactive Measurement questions will then be skipped, leaving the responses in their default or previously defined states.

If this question is answered “yes”, the following series of questions will be presented in sequence.

(a) PORT 1?    off    (drive)

The “drive” option causes the internal analysis unit to output a pulse to Port 1 when the analysis trigger is encountered. This function is useful for arming or triggering an external measurement instrument such as a scope or logic analyzer.

If “off” is selected, PORT 1 has no function.

(b) PORT 2?    off    (drive)

The “drive” option causes the internal analysis unit to output a pulse to Port 2 when the analysis measurement is complete. This function is useful for arming or triggering an external measurement instrument such as a scope or logic analyzer.

If “off” is selected, Port 2 has no function.

(c) Active edge?    rising    (falling)

This question is only encountered if either Port 1 or Port 2 is configured to operate in the “drive” mode. The response specifies the polarity of the drive pulse which will be generated at the active ports.

“Rising” specifies a positive going output pulse whereas “falling” specifies a negative going output pulse. The polarity specification applies to both ports if both are active.

The following questions refer to the lines available through the IMB connector on the internal analysis board, and on other interacting modules.

(d) Trigger enable?    off    (drive) (receive)

1. No IMB Interaction over the trigger enable line.

If the “off” option is selected, internal analysis will not interact with the trigger enable line.

2. Drive IMB Trigger Enable

Selection of the “drive” option causes internal analysis to drive the IMB trigger enable line when analysis finds the internal trigger point or receives an external trigger.

### 3. Receive IMB Trigger Enable

Selection of the “receive” option prevents internal analysis from finding its internal trigger point until some other module has driven the trigger enable line.

The trigger enable options are the only IMB functions available when using the 40 channel (64300A) internal analysis board. With the 48 channel (64302A) board the following additional options become available:

For 48 channel analysis there is one function that is always used whenever any other interaction is desired. This is the function of receiving the IMB Master line in order to allow synchronous initiation of the multiple modules. Internal analysis will select the correct option for this function depending on the options chosen for the other functions.

(e) External trigger?    off    (drive) (receive) (drive and receive)

#### 1. No interaction over IMB trigger line.

When “off” is selected, internal analysis will not participate in any interaction over the IMB trigger line.

#### 2. Drive IMB trigger

Selection of the “drive” option causes internal analysis to drive the trigger line when it finds its internal trigger point.

#### 3. Receive IMB trigger

Selection of the “receive” option allows internal analysis to trigger either on finding its internal trigger point or when another module drives the IMB trigger line.

#### 4. Drive and receive IMB trigger

Internal analysis will search until it finds its internal trigger or until another module drives the trigger line. Regardless of the source of the trigger, once internal analysis has triggered, it begins to drive the IMB trigger line.

(f) Internal trigger?    on    (off)

1. Enable internal trigger

If the “on” option is selected the internal triggering mechanism is enabled. This means that triggers specified via a “trace” or “specify trace” command will cause internal analysis to trigger if they are enabled (see trigger enable option above).

2. Disable internal trigger

If the “off” option is selected, then the internal triggering mechanism is disabled and will not cause a trigger. Thus triggers specified by “trace” or “specify trace” command will be ignored and internal analysis will only trigger when it is receiving an external trigger.

(g) Delay clock?    off    (drive)

1. No interaction on delay clock line

If the “off” option is selected then internal analysis will not interact over the delay clock line.

2. Drive delay clock line

Selecting the “drive” option causes internal analysis to drive the delay clock line once it has triggered, whether by an internal trigger or a received external trigger.

# Chapter 8

## Simulated I/O

### Introduction

The “Simulated I/O” feature of the 64000 System allows the user to develop programs for, without actually using, the target system's I/O hardware. To do this, the 64000 system's I/O hardware is used to “simulate” the target system's I/O hardware. This provides a double benefit. First, programs may be developed concurrently with hardware development, and second, if the target systems hardware exists but is not available to the programmer, program development can continue uninterrupted.

The following 64000 system hardware may be used to “simulate” the target system hardware during user-program development. (The 64000 hardware is listed in the order of description.)

- . Printer
- . Display
- . Keyboard
- . Disc
- . RS-232 Communications Channel

Simulated I/O is described in this section as follows. First an overview is presented. The overview describes the common attributes of the five simulated I/O interfaces, and then briefly, the interfaces themselves. The intent of the overview is to acquaint the reader with the simulated I/O features.

Following the overview, each interface is described in detail. The intent of the detailed descriptions is to provide sufficient information to allow a user to write the programs that will interface with the 64000 I/O devices. Following the detailed descriptions is a list of error codes, sample programs and file formats.

After the I/O programs have been written, assembled or compiled, and linked, they may be incorporated into an emulation configuration, then executed and tested.

Emulation configuration is described in Chapter 4 of this manual. Running and testing the programs is done with the commands described in Chapters 4 thru 7 of this manual.

## **Overview**

A general description of each of the simulated I/O interfaces is described in the following paragraphs. However, all of the interfaces have common attributes. These are described first.

## **Common Attributes**

Each simulated I/O interface requires a unique memory location to which all I/O handshaking codes are sent by both the user and the 64000 programs. The address for this location is generically referred to as the control address, or CA. The 64000 samples these addresses periodically looking for commands. Location CA must be initially defined in the users program and in the emulation configuration. If more than one simulated I/O interface is to be implemented, then the user must make sure that each I/O program assigns a unique address for the CA. Additionally, the user program must allow for contiguous buffer spaces following the CA. The exact amount, and use, of this buffer space is determined by the type of I/O interface. These requirements are specified in the detailed descriptions of the interfaces.

The addresses for the different CA locations are entered into the 64000 program during emulation configuration. The processor must not be restricted to real time runs when using simulated I/O. The CA locations must be located in memory space assigned as either user RAM or emulation RAM. It is recommended that the CA locations be in emulation RAM since this will allow the user programs to run faster. Mapping the CA locations to user RAM will cause the emulator to go to the monitor program while polling the CA locations for commands and/or data.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed in locations CA+1 thru CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

The user program must initiate the request to open the simulated I/O interface. To do this, after setting up the supplemental information in locations CA+1 thru CA+n, the user program places the appropriate code into location CA. (Code 80H opens all interfaces except the disc file where it creates a file.) If the 64000 program successfully executes the request, it returns the appropriate code to location CA. (Usually a 00 is returned, but not always.) If the 64000 program cannot execute the request, an error code is returned to location CA. A group of predefined error codes is used. Within this group only a portion of the codes apply to each interface. These error codes are defined in general terms in Table 8-8 which is located toward the end of this chapter. For those interfaces where the error codes also have specific meanings, the meanings are defined in the detailed descriptions of the interface. When the user is finished with the system resources, he should "close" the appropriate interfaces with the proper commands. All devices will automatically be closed by an "end" command or by execution of a reset-reset.

### **Printer I/O Interface (See Figure 8-1)**

This is the simplest of the five I/O interfaces. Only three user-control codes are used to interface with the printer. These are: (1) open printer file, (2) write to the printer, and (3) close printer file.

A buffer space contiguous to location CA contains a value indicating the number of bytes (characters) to be printed followed by the characters themselves.

### **Display I/O Interface (See Figure 8-2)**

This is somewhat more complex than the printer I/O interface since it has five user control codes. These codes are used to: (1) open the display file, (2) roll to and write line 18 (this is used to scroll lines up on the display), (3) select a starting line and column, (4) write from the selected line and column, and (5) close the display.

Depending upon the control code issued, a buffer space contiguous to location CA is required to hold one of the following parameter groups: (1) line length in bytes followed by the bytes to be displayed, (2) line and column number at which record display is to begin, or (3) record length in bytes followed by the record bytes to be displayed. The open and close codes use no additional buffer space other than location CA.

### **Keyboard I/O Interface (See Figure 8-3)**

The keyboard interface uses two user control codes and two keyboard input command word codes. Additionally, the 64000 returns one of 24 keyboard output command word codes.

The user control codes are used to open or close the keyboard interface file. The two keyboard input command codes are used to either: (1) clear the currently displayed line upon receipt of a keyboard character, or (2) append the character to the existing line.

When the keyboard file is opened, a buffer space contiguous to location CA is required to hold the keyboard input command word and the maximum record length specification. This specification defines the maximum record length that will be accepted from the keyboard. Thus, the buffer must be large enough to accept the keyboard output parameters and the maximum record length specified.

The keyboard output command word defines the manner in which the input line was terminated or the status of the keyboard output record. The output record consists of ASCII coded character bytes.

## Disc Files I/O Interface (See Figure 8-4)

### CAUTION

---

The disc file simulated I/O control codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Incorrectly accessing these files may destroy them and cause serious system problems!

---

The simulated disc file interface uses ten user control codes. These codes allow the user program to: (1) create, open, close, or delete a file; (2) advance to, backup to, or randomly select a record position within a file; (3) automatically select record position 1 in the file; and (4) read from, or write into any selected record position in the file. The user may also assign a different file name to be associated with an already existing CA.

Depending upon the control code issued, a buffer space contiguous to location CA is required to hold one of the following parameter groups: (1) file type number, (2) disc number, (3) record number, (4) maximum number of words to read or write, or (5) the actual number of words read or written, followed by the words themselves.

No buffer space is required following the control codes used to close the file and to automatically select record position 1 in a file.

## RS-232 I/O Interface (See Figure 8-5)

This is the most complex of the five I/O interfaces. To use this interface, the following distinct events **MUST** be implemented between the user and 64000 programs: (1) the RS-232 interface must be opened; (2) the 8251 Universal Synchronous/Asynchronous, Receiver/Transmitter, or USART, is initialized; (3) using the appropriate command word, an 8251 operating mode is selected; (4) data may be written to, or read from, the 8251; and (5) when data transfer is complete, the RS-232 file may be closed.

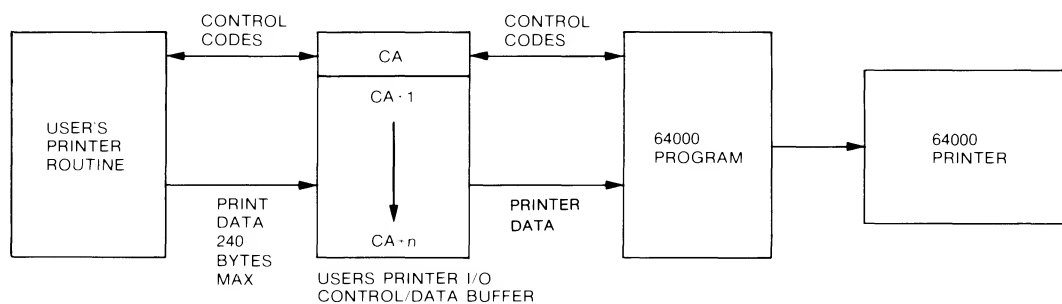
To implement the interface, the user program must allow for control space contiguous to location CA as shown in Figure 8-5. During 8251 initialization, locations CA+1 thru CA+5 hold the command and status words used to initialize and select the operation of the 8251.

The user program may read or write single bytes or multiple-byte records. When reading or writing single bytes, the single byte is passed through location CA+1. If multiple byte records are to be handled, the user program must set up read and write buffers as shown in Figure 8-5.

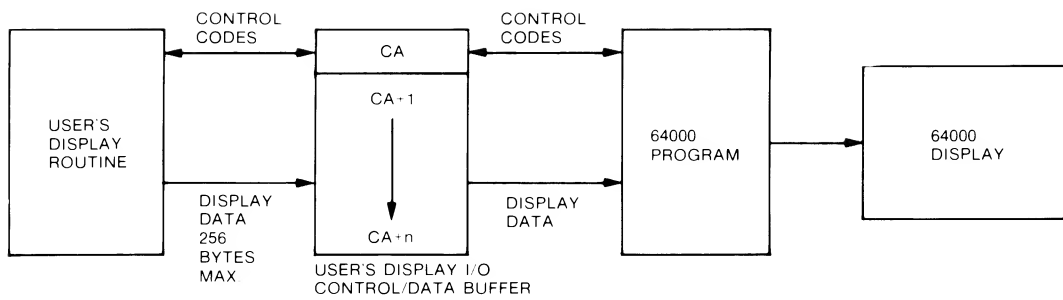
When writing multiple byte records, locations CA+6 thru CA+22 hold the write buffer pointers and the actual number of bytes sent by the 8251. This data is used interactively between the user and 64000 programs to transfer write data from the users program, via the users and 64000 write buffers, to the 8251.

When reading multiple-byte records, location CA+23 thru CA+39 hold the read buffer pointers and the actual number of bytes received by the 8251. This data is used interactively between the user and 64000 programs to transfer read data from the 8251, via the 64000 and users read buffers, to the user program.

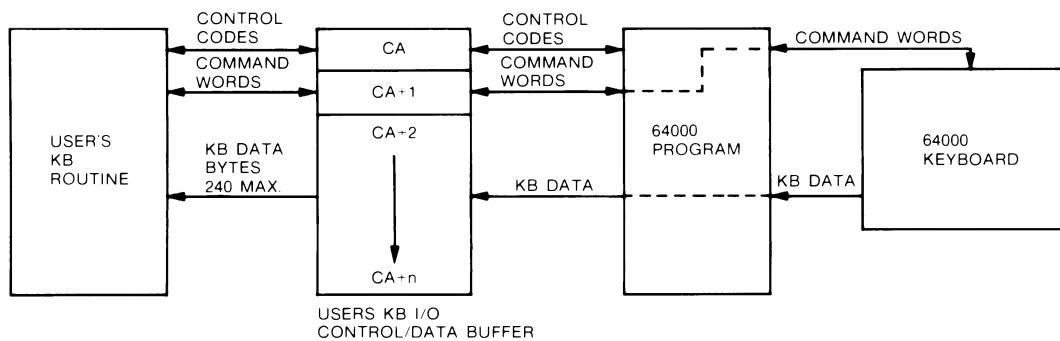
The read and write buffers may be updated separately or together by the user program.



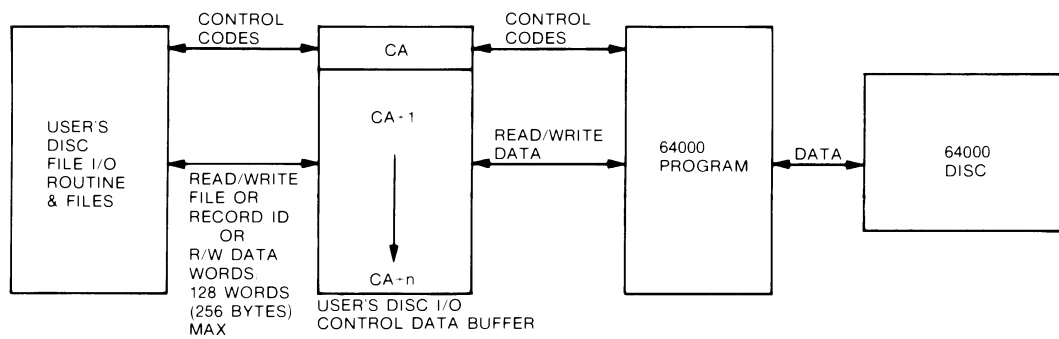
**Figure 8-1. Simulated Printer I/O Interface Diagram**



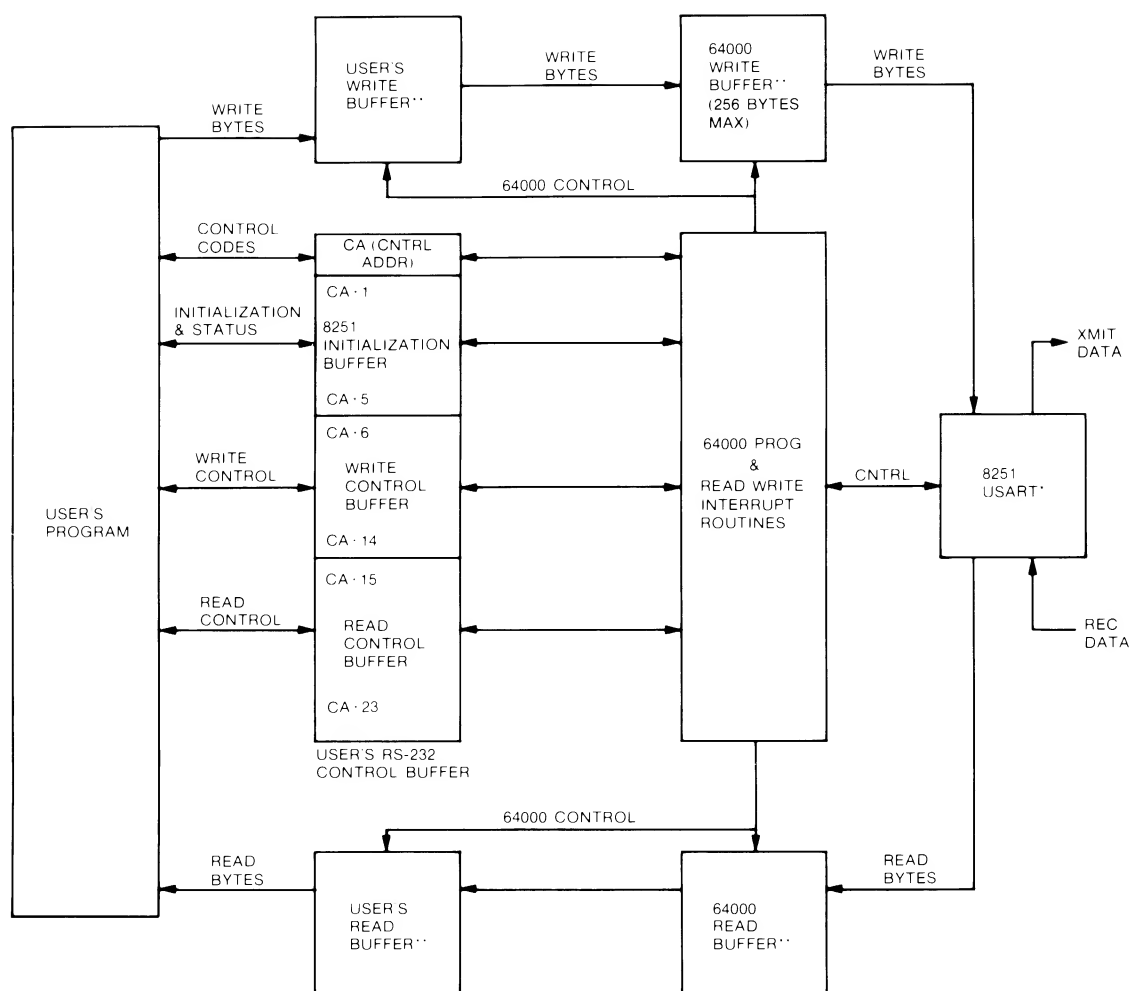
**Figure 8-2. Simulated Display I/O Interface Diagram**



**Figure 8-3. Simulated Keyboard I/O Interface Diagram**



**Figure 8-4. Simulated Disc File I/O Interface Diagram**



\*USART = Universal Synchronous/Asynchronous Receiver/Transmitter.

\*\*Buffers are required only if records are to be read or written. Single bytes do not require these buffers.

**Figure 8-5. Simulated RS-232 I/O Interface Diagram**

# Printer I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 program for printer I/O to occur. The events are:

- . Open Printer File
- . Write to Printer
- . Close Printer File

The above events, the corresponding control codes, and parameters, where applicable, are summarized in Table 8-1.

## NOTE

---

During the time that a simulated I/O printer file is open, no other user can access the printer. Thus, be sure to close the file when finished.

---

## Open Printer (80H)

Before using a "write to printer" code, the user program must request that the printer interface be opened. This is done by placing code 80H into location CA.

## NOTE

---

CA represents the memory location to which all printer I/O "handshaking" codes are sent by both the user and the 64000 program. The actual address for the printer is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - printer, RD-232, dispaly, etc. - requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is generally contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed into locations CA+1 thru CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

---

The 64000 program responds by opening the printer file and returning a 00 to location CA. If the file cannot be opened, error codes are returned as shown in Table 8-1.

After the file is opened, the user program may issue a write-to-printer code as described in the next paragraph.

### **Write to Printer (82H)**

To send a write record to the printer, the user program places the following parameters into locations CA+1 thru CA+n and then after setting up locations CA+1 thru CA+n, places code 82H into location CA.

The record length in bytes is entered into location CA+1. The record length must be a minimum of two bytes and may be a maximum of 240 bytes in two byte increments. That is - the record must always contain an even number of bytes. Odd bytes should be padded with a space (20H).

Locations CA+2 thru (CA+2)+n contain the ASCII codes of the character to be printed.

The 64000 responds by supplying the write record to the printer and returning a 00 to location CA. The 64000 automatically sends a carriage return/linefeed to the printer following the user data. If the write-to-printer record is not accepted, an error code is returned as listed in Table 8-1.

### **Close Printer File (81H)**

The user program closes the printer file by placing code 81H into location CA. The 64000 responds by closing the file and returning code 00 to location CA. The 64000 will perform a form feed automatically.

If the close file is not accepted, an error code is returned to location CA as shown in Table 8-1.

**Table 8-1. Printer I/O Codes**

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN PRINTER FILE	CA	80H	CA	00	01 thru 08
					09: file is already open.
					10-14: NA
CLOSE PRINTER FILE	CA	81H	CA	00	01 thru 08
					09: file is already closed.
					10-14: NA
WRITE TO PRINTER	CA	82H	CA	00	01 thru 08
	CA+1	Record Length in bytes (240 max.)	The 64000 accepts the record and causes it to be printed.		09: file is not open.
					10, 11, 13 & 14: NA
	CA+2	Record byte 1*			12: Record length exceeded 240 bytes.
	↓	↓			
	(CA+2) +n	Record byte n*			

\*All display characters must be formatted in ASCII code. A code greater than 0F0H will not be accepted by the 64000 program.

NA= Not Applicable.

See table 8-8 for complete error code listing.

# Display I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 programs for display I/O to occur. The events are:

- . Open Display File
- . Roll To / Write line 18 (scroll and write)
- . Select line and column
- . Write from selected line/column
- . Close Display File

The above events, the corresponding control codes and parameters, where applicable, are summarized in Table 8-2. Display techniques are shown in Figure 8-6.

## NOTE

---

During the time that the simulated I/O display file is open, the standard 64000 keyboard has no control over the display.

To regain control, press the simulate softkey which closes the file. If the keyboard file is open, it, too, is closed when the softkey is pressed.

---

## Open Display File (80H)

Before any writing can be done on the display, the user program must request that the display file be opened. This is done by placing code 80H into location CA.

## NOTE

---

CA represents the memory location to which all display I/O “handshaking” codes are sent by both the user and the 64000 program. The actual address for the display I/O CA is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - display, RS-232, printer, etc. - requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is generally contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed into locations CA+1 thru CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

---

The 64000 program responds by opening the display file, and returning a 00 to location CA. If the file cannot be opened, error codes are returned as shown in Table 8-2.

After the file is opened, the user program may write on the display as described in the following paragraphs.

### **Roll To/Write Line 18 (82H)**

This command allows writing to be initiated at the bottom of the display. Sequential Roll Up/Write Line 18 commands cause the previously written line 18 to roll to line 17, etc. Thus, writing is always done on the bottom line and the previously written lines are shifted up as each new line 18 is written.

To cause the display to roll up and begin writing on line 18, the user program places the following parameters into location CA+1 thru CA+n, and after setting up locations CA+1 thru CA+n, then places code 82H into CA.

The line length in bytes is entered into location CA+1. The line length must be a minimum of two bytes and may be a maximum of 80 bytes, in two byte increments. That is, the line must always contain an even number of bytes. If the user writes an odd number of bytes, the 64000 will pad the line with a null.

Locations CA+2 thru (CA+2)+n contain the ASCII codes of the characters to be written on line 18. The 64000 responds by storing this data in a display buffer and returning a 00 to location CA. A delay may occur before the program rolls up and writes to line 18. Thus, a program wait may be required. If writing cannot be done, especially if write roll/column is used (roll/column does not use delay), an error code is returned as listed in Table 8-2.

After initially rolling up and writing on line 18, subsequent Roll Up/Write Line 18 commands cause the previously written line 18 to roll up to line 17, line 17 to roll to line 16, etc. Although the 64000 responds almost immediately with a 00 in CA, the actual scrolling of a line can take up to 200 msec. The 64000 will accept other commands during this time. Future scrolls are buffered and performed in sequence. Row/Column writes will be performed immediately and may be scrolled if a previous scroll has not been completed.

## Select Starting Line/Column (83H)

The user programs may specify the line number and column number at which writing, when indicated, will start. To do this, the user program places the line number (1 thru 18) into location CA+1, the column number (1 thru 80) into location CA+2, and then places code 83H into location CA.

The 64000 responds by storing the line and column number and returning code 00 to location CA. The line and column numbers are stored until either writing is initiated (code 84H) or the display file is closed.

If the line and column numbers are not accepted by the 64000 program, an error code is returned to location CA as listed in Table 8-2.

Figure 8-6 shows the display techniques.

## Write From Starting Line/Column (84H)

Before writing can be initiated, a starting line number and column number must be specified by the user program. After this is done, writing may be initiated as follows: the user program initiates writing by placing the record length (i.e., number of characters to be displayed) into location CA+1, the actual display characters (ASCII codes) into locations CA+2 thru (CA+2)+n, and then places code 84H into location CA.

The maximum record length is 255 bytes. The display characters must be formatted in ASCII codes. The 64000 program will not accept a display code greater than 0F0H.

The 64000 responds by displaying the record beginning at the starting line and column specified by code 83H. If the record exceeds the length of the starting line, writing continues at column one of the next line, etc.

If the 64000 cannot initiate writing as requested, an error code is returned to location CA as shown in Table 8-2.

## Close Display File (81H)

The user program closes the display file by placing code 81H into location CA. The 64000 responds by closing the file and returning code 00 to location CA.

If the close file is not accepted, an error code is returned to location CA as shown in Table 8-2.

Pressing the inverse video "simulate" key or performing a "reset-reset" will automatically close the display. Closing the display also closes the keyboard.

**Table 8-2. Display I/O Codes**

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN DISPLAY FILE	CA	80H	CA	00  The 64000 program opens the file and clears the display	01 thru 08 & 14
					09 code >84H or file is open
					10 thru 13: NA
CLOSE DISPLAY FILE	CA	81H	CA	00	01 thru 08 & 14
					09: file is already closed.
					10 thru 13: NA
ROLL TO/ WRITE LINE 18	CA	82H	CA	00	01 thru 08 & 14
	CA+1	Line length in bytes (80 max)	The 64000 program stores this data in a display buffer. A delay may occur before rolling to and writing on line 18 actually occurs. A program wait may be required. If successive line 18's are written, then the preceeding line 18 is rolled to line 17, 17 to 16, etc.		09: file is not open
	CA+2	Line byte 1*			10, 11, & 13: NA
					12: Invalid record length
	(CA+2) +n	Line byte n*			

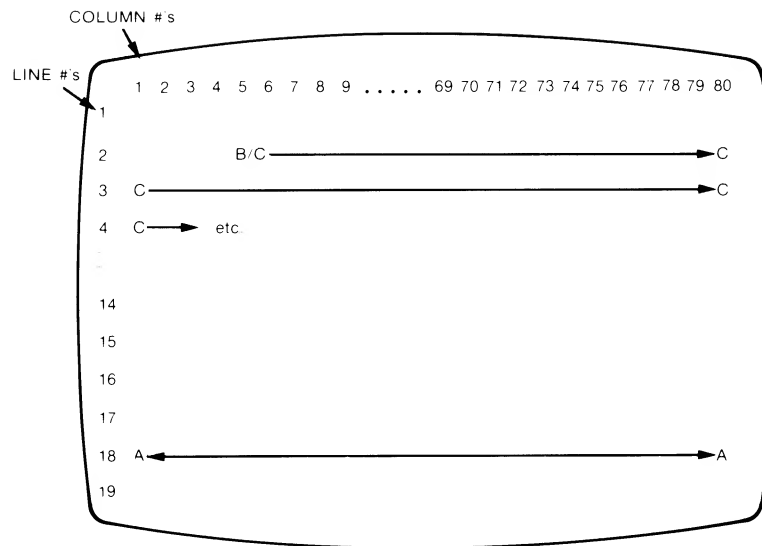
**Table 8-2. Display I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
	Request		Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
SELECT STARTING LINE/ COLUMN	CA	83H	CA	00	01 thru 08 & 14
	CA+1	Line # (1-18)	The 64000 program stores the line and column numbers until a write line/column request is issued or the file is closed.		09: File is not open
	CA+2	Column Number (1-80)			10, 12 & 13: NA  11: Invalid line or column number.
WRITE FROM STARTING LINE/ COLUMN	CA	84H	CA	00	01 thru 08, 13 & 14
	CA+1	Record length in bytes (255 Max)	The 64000 program displays the record starting at line/column selected by code 83H. If record exceeds one line, writing continues at column 1 of next line,etc. See figure 8-6.		09: file not open.  10 & 12: NA
	CA+2 ↓ (CA+2) +n	Record byte#1 ↓ Record byte n*			11: line/column not specified by 83H.

\*All display characters must be formatted in ASCII code. A code greater than 0F0H will not be accepted by the 64000 program.

NA= Not Applicable.

See table 8-8 for complete error code listing.



**64000 DISPLAY**

DISPLAY  
LETTER

MEANING

- A Code 82H automatically causes the display to roll to line 18. Up to 80 characters, in two byte increments, may be written on the line. Sequential Roll To / Write Line 18 commands cause the previous line 18 to roll to line 17, line 17 to roll to line 16, etc.
- B/C B is the point (line 2, column 5) defined by code 83H at which writing will begin. C is the statement which is defined by code 84H and begins at point B. There is no limit on the record length defined by 84H. If the record exceeds the length of line 2, it is continued on line 3 at column 1, etc.

**Figure 8-6. Display Techniques**

# Keyboard I/O Interface

The operation of the keyboard I/O interface is described in the following four phases:

- . User Program Requests Keyboard Read
- . 64000 Response to Keyboard Read Request
- . 64000 Detects Positive KB Output Command Word
- . User's Program Detects 00 in CA

Each of the above phases corresponds to a significant interaction which must be implemented between the user program and the 64000 program for keyboard I/O to occur.

The keyboard I/O interface events are summarized in Figure 8-7 and Table 8-3.

## NOTE

---

To automatically close the simulated I/O keyboard file and return the keyboard to standard operation, press the **simulate** softkey. If the display file is also open, it, too, is closed when the softkey is pressed.

---

## User Program Requests Keyboard Read (80H)

Before any other keyboard operation can be initiated, the user program must request that the KB I/O interface be opened. This is done by first placing the KB-input-command word and the maximum record length specification into the KB I/O buffer as shown in Phase I of Figure 8-7. Then, after setting up locations CA+1 thru CA+n, code 80H is placed into location CA of the buffer.

## NOTE

---

CA represents the memory location to which all KB I/O codes are sent by both the user program and the 64000 program. The actual address of CA is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - keyboard, RS-232, printer, etc., requires its own unique interface.

Certain I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed into locations CA+1 thru CA+n BEFORE the corresponding control code is placed into CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

---

The KB-input-command word is placed in buffer location CA+1. This word contains either a “-1” or “-2” code. A “-1” code causes the current line not to be cleared on the first character (i.e., the current keyboard characters are appended to any characters already displayed on the same line). A “-2” code causes the current line to be cleared on the first character (i.e., previously displayed characters are erased from the line and only the current keyboard characters are displayed).

The maximum record length specification is placed in buffer location CA+2. This is the maximum record length (i.e., number of keyboard characters) that the user program will accept from the keyboard. The record length specification may specify up to 240 characters (3 lines on the 64000 display). However, the keyboard may transmit more or less characters than this specification. If the number of characters transmitted exceeds the record length specification, the user program is informed of this by an applicable code in the KB-output-command word as described below.

### **64000 Response to Keyboard Read Request**

The 64000 program responds to the KB read request by storing the KB-input-command word and record length specification, and by placing code 82H into location CA as shown in Figure 8-7.

The 64000 program sets the KB-output-command word to the same code specified in the KB-input-command word (-1 or -2).

The 64000 then begins monitoring the keyboard until an output command word is detected. The result of this detection is described in the following paragraphs.

### **64000 Detects Positive KB-Output-Command Word**

The keyboard may send either a KB-output-command word by itself or a command word followed by one or more keyboard characters. In either case, when a KB-output-command word is detected, the 64000 program places the word, and if applicable, other data into the KB I/O buffer as shown in Figure 8-7 (Phase III). The KB output word, which is always sent, is placed in buffer location CA+1.

The 64000 program places a 00 in location CA to indicate to the user program that either a KB command and/or data is now available.

If keyboard characters are also sent and if a “lost character” was generated, then the “lost character” is placed into location CA+2. (How a “lost character” is generated is described later.) Also, when keyboard characters are sent, the actual number of characters in the string (i.e., actual record length) is placed into location CA+3. The keyboard characters themselves (ASCII coded bytes) are placed into locations CA+4 thru (CA+4)+n.

The KB output command in location CA+1 may be any one of the codes shown in Table 8-4. Two of these codes, 8 and 24, will occur only if the actual record length from the keyboard exceeds the maximum record length specification. If either of these codes is generated, then location CA+2 contains the ASCII code of the surplus or lost character that exceeded the specified record length. A lost character may be generated in either of two ways:

- a. When characters are entered as a continuous string and the string exceeds the specified record length. For this case, the first character to exceed the specified record length is placed in “lost character” location CA+2. If typing continues, each individual surplus character is placed into the “lost character” location CA+2 replacing the previous character. Thus, the last “lost character” entered remains in location CA+2.
- b. When a character is inserted into a full record. For this case, the character at the end of the already full record is placed into “lost character” location CA+2. If additional characters are inserted, each succeeding end character is placed into CA+2, replacing the previous character.

### **User's Program Detects 00 in CA**

After detecting a 00 in location CA, the user program takes the data from the KB I/O buffer and places either 80H or 81H into location CA. The results of each of these response codes are as follows:

- a. 80H Response Code - Read Keyboard I/O

If the user program responds with code 80H, the KB-input-command word and record length specifications must be supplied by the user program as shown in Figure 8-7.

The 64000 program responds by again reading the keyboard.

- b. 81H Response Code - Close KB I/O

If the user program responds with code 81H, the 64000 program closes the KB I/O interface. This command will also close the display file if it was open.

**Table 8-3. Keyboard I/O Interface Codes**

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request Address	Valid User Request Contents	Invalid Request Error Code
OPEN KB INTER FACE	CA	80H		See 82H, below	08, 12, or 14
	CA+1	KB Input Command Word			Other codes do not apply
	CA+2	Max. Record Length Specification (up to 240 bytes)			
READ IN PROCESS		Initiated by 64000 program in response to 80H above	CA	82H 64000 stores KB-input-command word & max. record length spec. It then monitors KB-output-command word until positive word is detected and then responds as follows:	
OUTPUT AVAILABLE		Initiated by 64000 after 82H, above	CA CA+1	00 KB out-put command word	

Table 8-3. Keyboard I/O Interface Codes (Cont'd)

Request Name	64000 Response To:				
	User Program Request		Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
		User program may then respond to 00 with 80H or 81H as shown below.	CA+2	Reserved for Lost Character	
			CA+3	Actual record length (#of KB bytes)	
			CA+4 ↓ (CA+4) +n	KB Byte 0 ↓ KB Byte n	
CLOSE KB I/O	CA	81H	CA	00	08 or 14
					Other codes do not apply.

See Table 8-8 for complete error code listing.

**Table 8-4. Command Word Codes**

**Part A. KB - Input - Command Word**

<b>Code</b>	<b>Meaning</b>
-1	Current line not cleared. Characters appended to previously displayed characters.
-2	Current line cleared. Previously displayed characters erased.

**Part B. KB - Output - Command Word**

<b>Code</b>	<b>Meaning</b>
8	Insert character in full line (lost character placed in CA+2)
9	Tab Key
10	Down arrow key
11	Up arrow key
12	Display next page
13	Carriage return
14	Attempting to move cursor right past last allowed screen location
15	Attempting to move cursor left past first allowed screen location
16	Delete character from full line
17	Shift key
18	Display previous page
19	Roll display down
20	Roll display up
21	Shift right arrow key
22	Shift left arrow key
23	Clear line key
24	Actual record length exceeded record length specification (lost character placed in CA+2)

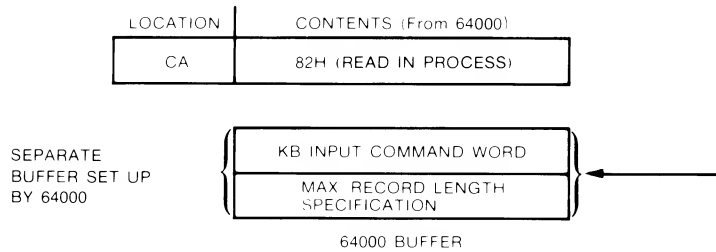
## Phase I - User Requests Interface Opening

LOCATION	CONTENTS (From User Program)
CA*	80H (OPEN KB I/O)
CA+1	KB INPUT COMMAND WORD
CA+2	MAX. RECORD LENGTH SPECIFICATION (UP TO 240)

KB I/O BUFFER

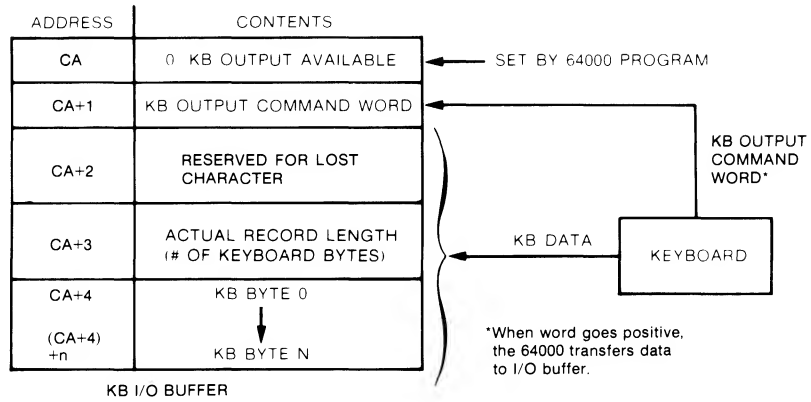
\*The actual address for location "CA" is defined by the user during configuration of the emulation "CMDFILE"

## Phase II - 64000 Response to Open-Interface Request



**Figure 8-7. Keyboard I/O Interface Sequence**

### Phase III - 64000 Detects Positive KB Output Command Code



Phase IV - The user program may respond with either an 80H code as shown for phase I or an 81H code which closes the simulated keyboard I/O interface.

**Figure 8-7. Keyboard I/O Interface Sequence (Cont'd)**

# Disc File I/O Interface

**CAUTION**

---

The disc file simulated I/O control codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Incorrectly accessing these files may destroy them and cause serious system problems!

---

The following paragraphs describe the type of files and the events which must be implemented between the user and the 64000 program to either: (1) create a new disc file, or (2) read from, write into, delete, or change the name of an existing file. The file types are described first. Then, the program events are described in the following order:

a. Creating New File

- 1) Creating File (80H)
- 2) Writing First Record (89H)
- 3) Writing Additional Records (89H)
- 4) Closing Created File (82H)

b. Accessing Existing File

- 1) Opening File (81H)
- 2) Selecting Record
  - (a) Automatic selection of records 1, 2, 3, ... etc.
  - (b) Advance "N" records (84H)

- (c) Backup "N" records (85H)
- (d) Position to record "N" (86H)
- (e) Rewind to record one (88H)
- 3) Reading Record (87H)
- 4) Writing Record (89H)
- 5) Closing Open File (82H)
- c. Deleting File (83H)
- d. Changing File Name Associated with a CA (8AH)

The predefined file types are listed in Table 8-5.

Table 8-6 summarizes the user program requests, the corresponding control codes, and, where applicable, corresponding parameters.

## File Types

The names and type numbers are listed in Table 8-5.

## Creating New File

### Creating File

To create a new file, the user program places the file type number into location CA+1, the disc number into location CA+2, and then places code 80H into location CA. (The disc number is the disc upon which the file will reside.)

### NOTE

---

CA represents the memory location to which all disc file I/O "handshaking" codes are sent by both the user program and the 64000 program. The actual address for the disc files CA is defined in the user program and entered into the 64000 during the configuration of the emulation CMDFILE. Each I/O interface - disc files, display, keyboard, etc. - requires its own unique CA address.

Certain I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed into locations CA+1 thru CA+n BEFORE the corresponding control code is placed into CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

---

The 64000 responds by creating the file type requested and returning a 00 to location CA which indicates the file has been created.

If the file cannot be created, an error code as shown in Table 8-6 is returned to location CA. (General definitions for the error codes are listed in Table 8-8.)

After the file is created, the user program may either write records immediately into it, or close it, and then reopen it and write records into it later.

#### **Writing First Record**

After a file is created the first record is written into it as follows. The user program places parameters, as described below, into locations CA+1 thru CA+n, and then places code 89H into location CA.

The number of words in the write record is placed into location CA+1. A write record may contain up to a maximum of 128 words (256 bytes). Thus, an even number of bytes (whole words) must always be written.

Locations CA+2 thru (CA+2)+n contain the words of the write record.

The 64000 responds by automatically writing the records into the file as record number 1. After the record is successfully written, the 64000 returns a 00 to location CA. If the record cannot be written, an error code, as listed in Table 8-6, is returned to location CA.

Additional records are written into the file as described in the next paragraph.

#### **Writing Additional Records**

If the newly created file is still open (i.e., has never been closed), additional records are written into the file as described for record one with the following difference. Each succeeding record is automatically written with the next corresponding record number. Thus, the second record written becomes record number 2, the third record written becomes record number 3, etc.

### **Closing Created File**

To close the newly created file, the user program places code 82H into location CA. The 64000 responds by closing the file and returning a 00 to location CA. If the file cannot be closed, an error code, as listed in Table 8-6, is returned to location CA.

## **Accessing Existing Files**

### **Opening File**

To open an existing file, the user program places the file type number into location CA+1, the disc number into location CA+2, and then placed code 81H into location CA.

The 64000 responds by opening the file and returning a 00 to location CA which indicates the file is open. If the file cannot be opened, an error code, as shown in Table 8-6, is returned to location CA.

### **CAUTION**

---

When a record is written into a file, it always becomes the last record in the file. Thus, writing a record into any location other than at the end of the file effectively erases all the following records in the file. When accomplishing the following paragraph choose record positions with care!

---

After the file is opened, the user program may either: (1) immediately read/write record 1, (2) select any record for reading, or (3) select a position within the file to begin writing.

### **Selecting Record**

Records are selected in any of the following ways:

- a. Automatic selection of records 1, 2, 3, ..., etc. When the file is opened, record 1 is automatically selected. Thus, it may be immediately written into, or read from, without first selecting it with an "advance", "position", or "rewind" code. After reading or writing record 1, record 2 is automatically selected and may be read from, or written into. This process can be continued for records 3, 4, 5, ..., etc.

### **NOTE**

---

Remember, that when a record is written into a file, it becomes the end of the file.

---

- b. Advance "N" Records. Records located ahead of the currently selected record (i.e., those records with higher numbers) may be selected as follows. The user program places the number of records into locations CA+1 and CA+2, and then places code 84H into location CA. The number of records is selected with a 15-bit word. The eight least significant bits are located in CA+1. The seven most significant bits are located in CA+2. The most significant bit in CA+2 is not used.

The 64000 responds by advancing the specified number and returning a 00 to location CA. If the record cannot be selected, an error code, as shown in Table 8-6, is returned to location CA.

After the record is selected, the user program may then either read from or write into it.

- c. Backup "N" Records. Records located behind the currently selected record (i.e., those records with smaller numbers than the current record) are selected in a way very similar to "advance 'N' records". The only difference is that backup code 85H is placed into location CA. Locations CA+1 and CA+2 contain the number of records as defined in subparagraph b above. The 64000 also responds as described above.
- d. Position to Record "N". Any record within the file may also be selected without knowing its location relative to the current record. This method is also similar to the "advance" or "backup" methods. The difference is that position code 86H is placed into location CA. Location CA+1 and CA+2 contain the record number as defined in subparagraph b above. The 64000 responds as described above.
- e. Rewind to Record One. This is a fast way to select record 1. This method differs from the previous selection method in several ways. First, only record 1 can be selected using this method. Second, the user program places code 88H into location CA. Third, there are no entries required in locations CA+1 and CA+2. The 64000 program responds as described in subparagraph b above.

### **Reading Record**

Once a record has been selected by one of the methods described above, it may be read as follows. The user program places the maximum number of 16-bit words it will accept from the record into location CA+1. Up to 128 words may be accepted. (The recommended technique is always set CA+1 to 128. Then, after reading is complete, throw away those words not wanted, if any.) After specifying location CA+1, code 87H is placed into location CA.

If the record is read successfully, the 64000 responds as follows: code 00 is returned to location CA. The actual number of 16-bit words read from the buffer is placed in location CA+1. Location CA+2 thru (CA+2)+n contains bytes 0 thru n.

If the record cannot be read, an error code, as shown in Table 8-6, is returned to location CA.

### **Writing Record**

A new record may be written into an existing file in either one of two ways. The record may be added to the end of the file or it may be written over an existing record in the file. However, if an existing record is written over, then the newly written record becomes the last record in the file.

To add a record to the end of the file, the record selected must be one greater than the last record in the file. For example, if a file contains five records, then record 6 must be selected before writing is initiated. (If record 5 is selected, it will be written over by the new record.) After writing record 6, record 7 may be written by issuing another write code, etc.

To write over an existing record, first select the record and then initiate writing. Again, remember that all following records in the file are erased. For example, if a file contains 10 records, and record three is written over, then records four thru ten are erased.

### **Closing Open File**

An open file is closed in the same way as described for a newly created file. That is, the user program places code 82H into location CA. The 64000 responds by closing the file and returning a 00 to location CA. If the file cannot be closed, an error code, as listed in Table 8-6, is returned to location CA.

### **Deleting Files**

To delete a file, the user program places the file type into location CA+1, the disc number into location CA+2, and then places code 83H into location CA. The 64000 responds by deleting the file. If the file cannot be deleted, an error code is returned to location CA as shown in Table 8-6. This delete is similar to a "purge" command in the general operating system. The purged file does go into the recoverable file list.

### **Changing File Name Assigned to a Particular CA**

The file name associated with a given CA location may be changed. This does not rename any files on the disc, but simply changes the name in the emulation command file associated with a given CA. To do this the user must first make sure that the present file associated with the CA of interest is closed.

To change the file name in the emulation configuration file, the user program places the new name record into locations CA+1 thru CA+16, and then places code 8AH into location CA. The name record is a fixed length record consisting of eight, 16-bit words. This record contains the record name, USERID, and specifies the length of both of these items.

The name must contain at least one character and may be up to nine characters long. The ID may be up to six characters long. However, the name and ID lengths are specified in a unique way. Also, the words containing these characters must be packed in the name record. Specifying name and character lengths and packing the words are done in the same way as described for the "Microprocessor Configuration Record" in the Linker Symbols File description. This description is located toward the end of this chapter.

To actually change the name of an existing file, the user must copy the contents of the file under the old file name into the file with the new file name. Either one or both of these file names may be specified by the user program at run time and accessed after "change file name" has been issued to the appropriate CA locations.

**Table 8-5. Disc File Type Numbers and Names\***

<b>File Type Number</b>	<b>File Name</b>
2	Source
3	Relocatable
4	Absolute
5	Listing
6	Emulation Command
7	Linker Command
8	Trace
10	Data
12	Assembler Symbols
13	Linker Symbol
**14	Types are defined
thru	and numbers assigned
255	by user program.

\* Formats for selected files are described at the end of this chapter.

\*\* HP may require some unassigned numbers for future use. It is, therefore, strongly recommended that the DATA (type 10) file be employed for the user defined type file.

There are predefined types of files, identified by numbers 2 thru 13, that may be created by the user program.

File type numbers 14 thru 255 may be assigned to files defined by the user program, as required. It should be noted, however, that HP may require some unassigned numbers for future use. It is, therefore, recommended that the user leave space for this possibility, starting with number 14.

### NOTE

---

Once created, file types 14 thru 255 can only be deleted by using the simulated I/O delete command.

---

The overall file name is assigned during emulation configuration. Under any one file name, only one each of a file type may be created. For example, a file named USA may only have one each of file types 2 thru 255. It cannot have two type 3 files.

### CAUTION

---

The disc file simulated I/O codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Incorrectly accessing these files may destroy them and cause serious system problems!

---

**Table 8-6. Disc File I/O Codes**

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request Address	Valid User Request Contents	Invalid Request Error Code
CREATE FILE	CA	80H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: file is not open
	CA+2	Disc #			11 thru 14: NA
OPEN FILE	CA	81H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: File is already open
	CA+2	Disc #			11 thru 14: NA
CLOSE FILE	CA	82H	CA	00	01 thru 08
					09: File is already closed
					10 thru 14: NA
DELETE FILE	CA	83H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: File not open

**Table 8-6. Disc File I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request Address	Valid User Request Contents	Invalid Request Error Code
ADVANCE "N" RECORDS	CA+2	Disc #			11 thru 14: NA
	CA	84H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
BACKUP "N" RECORDS	CA	85H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open.
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
POSITION TO RECORD "N"	CA	86H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
READ RECORD	CA	87H	CA	00	01 thru 08
	CA+1	Max. number of words user can accept. (128 words/ 256 bytes max.)	CA+1	Actual # of words read from buffer.	09: File is not open  12

Table 8-6. Disc File I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request Address	Valid User Request Contents	Invalid Request Error Code
			CA+2	Read Byte 1	10, 11, 13, 14: NA
			↓	↓	
			(CA+2) +n	Read Byte n *	
			(*256 bytes/ 128 words is max. record length.)		
REWIND TO RECORD ONE	CA	88H	CA	00	01 thru 08
					09: File is not open
					10 thru 14: NA
WRITE RECORD	CA	89H	CA	00	01 thru 08, 12
	CA+1	Number of words to be written. (128 words/ 256 bytes maximum.)			09: file is not open.
					10, 11, 13, 14: NA
	CA+2	Write byte 1			
	↓	↓			
	(CA+2) +n	Write byte n			

**Table 8-6. Disc File I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request Address	Valid User Request Contents	Invalid Request Error Code
CHANGE FILE NAME	CA	8AH	CA	00	01 thru 08 12 & 15
SEE NOTE BELOW		Bits 7-5 specify length of file name in 16-bit words-1. Bits 4 & 3 specify ID length in 16-bit words. Bits 2-0 contain all zeros. (See note below.)			09: File not open  10, 11, 13, 14: NA
	CA+2	First character of file name. Limited to capital letters A thru Z.			
	CA+3	Second and following file name characters may be small or capital letters,			

**Table 8-6. Disc File I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:	
	Address	Contents	Valid User Request Address	Invalid Request Error Code
		numerals 0 thru 9, underlines, and only if required one blank may be used to fill in last character in last word of name.		
	CA+4 thru CA+n. Where n 10	Up to 9 name characters may be used.		
	CA+ (n+1)	First USERID character.		
	CA+ (n+2) ↓ thru ↓ CA+16	Up to 6 USERID characters may be used.		
		See note below.		

Note: The name and USERID characters must be packed into a fixed length record. This record consists of 8, 16-bit words. Thus, the name record will always require a user buffer consisting of 17 bytes (byte CA through byte CA+16). All unused 16-bit words must be at the end of the record. No intervening unused words or bytes are allowed. If the last byte in the last name and ID word is not required to define the name, then it must contain an ASCII blank. The byte in buffer location CA+1 must be formatted the same as described for the most significant byte of word 16 in the name and user ID word block of the microprocessor configuration record. Refer to the "Microprocessor Configuration Record" in the Linker Symbols description for more information.

## RS-232 I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 programs for RS-232 I/O to occur.

These events are:

- o Open RS-232 File
- o Initialize 8251
- o Command To 8251
- o Status From 8251
- o Write To 8251

Write Single Byte

Write Record

- o Read From 8251

Read Single Byte

Read Record

- o Updating Read/Write Buffers

The above events, corresponding control codes, and parameters, where applicable, are summarized in Table 8-7.

### Open RS-232 File (80H)

Before any other RS-232 operation can be initiated, the user program must request that the RS-232 File be opened. This is done by placing code 80H into location CA.

#### NOTE

---

CA represents the location to which all RS-232 I/O “handshaking” codes are sent by both the user and the 64000 programs. The actual address for the RS-232 CA is defined in the users program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - RS-232, display, printer, etc.- requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 thru CA+n. The supplemental information must be placed into locations CA+1 thru CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 thru CA+n.

---

The 64000 responds by opening the RS-232 file and returning a 00 to location CA to indicate that the file is open. If the file cannot be opened, error code 08 or 09 is returned to location CA.

After the file is opened, the 8251 must be initialized as described in the next paragraph.

### **Initialize 8251 (82H)**

In general, 8251 initialization consists of resetting the 8251 and then selecting one of the following three operating modes: (1) asynchronous, (2) synchronous with one sync character, or (3) synchronous with two sync characters. (See Figure 8-8.)

For each of the three modes, the user program requests initialization by first setting up buffer locations CA+1 thru CA+5 and then placing code 82H into location CA. A command instruction with Internal Reset (IR) bit D6 set is placed into location CA+1. (See Figure 8-9.) The contents placed into locations CA+2 thru CA+5 depend upon the operating mode selected as described in the following paragraphs.

#### **Asynchronous Mode**

For this mode, the asynchronous mode instruction is placed into location CA+2 and a sync option word specifying 0 must be placed into location CA+3. Locations CA+4 and CA+5 contain no meaningful data.

The asynchronous mode instruction is used to select the baud rate, the character length, the parity parameters, and the number of stop bits. (See Figure 8-10.) (The only baud rates which may be used within the 64000 are the transmitter clock frequency ( $1 \times \text{Txc}$ ) or  $1/16 \times \text{Txc}$ . The baud rate factor of  $1/64 \times \text{Txc}$  cannot be used with the 64000. The basic frequency of  $\text{Txc}$  is selected by switches on the modem I/O card. Thus, the basic frequency ( $\text{Txc}$ ) may be changed by the I/O card switches.) The user must format this instruction so that the appropriate parameters are specified.  $1/16 \times \text{Txc}$  must be programmed if the baud rate is to match the baud rate Table in the Installation and Configuration Reference Manual.

The sync option specifies 0 since there are no sync characters for the asynchronous mode.

### **Synchronous Mode/Single Sync Character**

For this mode, the synchronous mode instruction is placed into location CA+2, the sync option word specifying "1" is placed into location CA+3, and the sync character is placed into location CA+4. Location CA+5 contains no meaningful data. (See Figure 8-8)

The synchronous mode instruction is used to select the character length, and the parity and synchronization parameters. (See Figure 8-11.) Bit D7 (SCS) of this word must specify a single sync character. The user must format this instruction so that the other appropriate parameters are specified.

The sync option word specifies "1" for a single sync character.

The format of the sync character must be defined by the user.

### **Synchronous Mode/Double Sync Character**

For this mode, the synchronous mode instruction is placed into location CA+2, the sync option word specifying "2" is placed into location CA+3 and sync characters 1 and 2 are placed into locations CA+4 and CA+5, respectively. (See Figure 8-8.)

The synchronous mode instruction is used to select the character length, and the parity and synchronization parameters. (See Figure 8-11.) Bit D7 (SCS) of this word must specify a double sync character. The user must format this instruction so that the other appropriate parameters are specified.

The sync option word specifies "2" for double sync characters.

The format of both sync characters must be defined by the user.

After the 8251 is initialized, the 64000 returns a 00 to location CA. If the 8251 cannot be initialized, error code 08 or 09 is returned as shown in Table 8-7.

## **Command to 8251 (83H)**

After the 8251 is initialized (i.e., reset and asynchronous or synchronous operation selected), it must be placed in the appropriate mode - transmit, receive, or combination transmit/receive, etc. To do this, the user program first places the appropriately formatted command word into location CA+1 and then places code 83H into location CA. (The user must format the command word to select the applicable operation as shown in Figure 8-9.)

The 64000 responds by supplying the command word to the 8251 and returning a 00 to location CA. If this cannot be done, code 08 or 09 is returned to location CA. (See Table 8-7.)

## **Status From 8251 (84H)**

The user may check the status of the 8251 at any time. To do this, code 84H is placed into location CA. The 64000 responds to this status request by returning a 00 to location CA and placing the 8251 status word in location CA+1.

The status word format is shown in Figure 8-12.

The status bits D0, D1, and D2 may be cleared or set by the 64000 program when operating in any of the buffered modes. If the user desires these bits to control operation, it is necessary to close the appropriate Tx or Rx buffers first.

## **Write To 8251**

The user program may write to the 8251 in either of two ways. It may write a byte at a time, or a write buffer may set up and data written continuously. Both methods are described. (Note: Before attempting to write data, the 8251 must be initialized and the command word, in the appropriate format, sent to the 8251 as described in the previous paragraphs.)

### **Write Single Byte (86H)**

To write a single byte to the 8251, the user program first places the write byte into location CA+1 and then places code 86H into location CA. (See Table 8-7.) The 64000 responds by supplying the byte to the 8251 and returning a 00 to location CA. If writing cannot be done, error code 08 or 09 is returned to CA. (See Table 8-7.) If more data is to be sent, it is recommended that the user poll the 8251 status to determine if the 8251 is ready to receive more transmit data.

### **Write Record (87H), Update Write Buffer (89H)**

(See also Update Read/Write Buffer (8DH)) - To write a record to the 8251, the user program must first set up a write buffer and identify the beginning and ending locations in the buffer. (The corresponding 64000 write buffer holds a maximum of 256 bytes.) (See Figure 8-13.) It then writes a record into the buffer and identifies the buffer locations into which the first and last bytes of the record are written.

The user program must then request that the record be transferred to the 8251. (See Figure 8-14.) This is done by first placing the user write buffers beginning/ending and first/last byte address pointers into locations CA+7 thru CA+22 and then placing code 87H into location CA.

The 64000 responds by transferring data from the users write buffer into a 64000 write buffer. (See Figure 8-15.) For each byte transferred to the 64000 buffer, the first byte address pointer (in locations CA+15 thru CA+18) is incremented by one. Data transfer continues until either all data in the users write buffer is transferred or the 64000 write buffer becomes full. (The 64000 write buffer holds a maximum of 256 bytes, or 128 words.) After a write buffer is set up and if update code 8DH or 89H is used, then the number of bytes actually transmitted by the 8251 is also entered into location CA+6 by the 64000 program. The number of bytes transmitted refers to the number of bytes transmitted from the 64000 buffer.

The user program should periodically examine the first and last address byte pointers (and if using update code 8DH or 89H, the number of bytes transmitted by the 8251 may also be examined) to determine the status of the buffer. (If the first and last byte pointers are equal, all data was transferred to the 64000 buffer.)

If all data was transferred, the user program may either supply another write record, or close the write buffer. If all data was not transferred, the user program may either wait until the remaining data is transferred, add more data to the buffer and update the last byte pointer, or close the write buffer. Each of these options is described in the following paragraphs.

Additional data may be added to, or a new record written into the buffer and the last byte address pointer updated as follows: If the first and last byte address pointers are pointing to the same location, the first new byte goes into the location pointed to by both pointers. If the first and last byte address pointers are not pointing to the same location, then the first new byte goes into the location just ahead of the one pointed to by the last byte address pointer (i.e., last byte address pointer + 1). Then the following bytes are entered into succeeding locations. (See Figure 8-15.)

After entering data into the buffer, the user program requests write data transfer. This is done by first placing the updated last byte address pointer into locations CA+19 thru CA+22 and then placing code 89H into location CA. (See Figure 8-16.)

The 64000 responds by transferring data from the users write buffer to the 64000 write buffer, increments the first byte address pointer for each byte transferred, and if update code 8DH or 89H is being used, the number of bytes sent by the 8251 is also updated.

Once the user program has placed code 8DH or 89H (update buffer) into location CA, the 64000 routinely monitors the last byte address pointer to determine if more data has been loaded into the users write buffer. If the 64000 detects that the last byte address pointer has been incremented, it transfers the data and increments the first byte address pointer to indicate the number of bytes written. It also updates the number of bytes sent by the 8251.

To write another record, the user program updates the last address pointer. The 64000 responds as described above.

To close the buffer, the user program places code 88H in location CA. The 64000 closes the write buffer and returns a 00 to location CA.

Data may be stored in the users write buffer using a “wrap around” method. That is, once the last location in the buffer is filled, the next byte is placed into the first location of the buffer. Thus, it is possible for the last byte address pointer to be pointing to an address that is less than (i.e., ahead of) the first byte address.

If any of the write buffer requests cannot be done, the 64000 returns the appropriate error code to location CA as shown in Table 8-7.

## **Read From 8251**

Reading data from the 8251 is similar to writing data to the 8251. The user program may read data in either of two ways. It may read a byte at a time or it may set up a read buffer and read a record at a time. Both methods are described. Note: Before attempting to read data, the 8251 must have been initialized and the command word, in the applicable format, sent to the 8251 as described in the previous paragraphs.

### **Read Single Byte (85H)**

To read a single byte from the 8251, the user program places code 85H into location CA. (See Table 8-7.)

The 64000 responds by returning a 00 to location CA and the read byte to location CA+1. If reading cannot be done, error code 08 or 09 is returned to CA.

The 64000 will return whatever character is in the Rx buffer of the 8251. It is recommended that the user check the status of the 8251 to see if Rx RDY is true before performing the single byte read. Any read operation will clear Rx RDY, indicating that the character in the buffer has been read.

### **Read Record (8AH) Update Read Buffer (8CH)**

(See also Update Read/Write Buffer (8DH)) - To read a record from the 8251, the user program must first set up a read buffer and identify the beginning and ending locations in the buffer. (See Figure 8-17.)

This is done by first placing the address pointers into locations CA+24 thru CA+39 and then placing code 8AH into location CA. Locations CA+24 thru CA+31 contain the address pointers for the beginning and ending locations of the users read buffer.

Locations CA+32 thru CA+39 contain the address pointers for the first and last bytes written into the buffer. These pointers are both initially set to point to the first location in the users read buffer. This indicates that the buffer is empty. (The 64000 will force the first data pointer to always point to the beginning of the buffer.)

The 64000 responds by continuously transferring read data from the 8251 to the 64000 read buffer. (See Figure 8-19.) The user program must then issue an 8CH or 8DH to transfer the data to the users buffer. For each byte transferred into the users read buffer, the last byte address pointer is incremented by one (see Figure 8-18). In addition, when update code 8DH or 8CH is being used, the number of bytes received by the 8251 and transfered into the 64000 is entered into location CA+23.

To determine when and how much read data is available, the user program must monitor the last byte address pointer and the number of bytes received. When read data is found in the buffer, the user program should process the data. If all data expected was received, the user program may then close the read buffer.

Once the user program has placed code 8CH or 8DH into location CA, the 64000 periodically monitors the output of the 8251, transfers data into the user read buffer, and updates the last byte address as required. The user program in turn monitors the last byte address pointer to determine if more data is available. This process continues until the user program closes the read buffer.

If code 8CH or 8DH is being used, and the user issues an 8AH again, the buffer is frozen for the user, yet the 64000 continues to receive data into its buffer.

To close the read buffer, the user program places code 8BH into location CA. The 64000 closes the buffer and returns a 00 to location CA.

Data may be stored in the user's read buffer using a "wrap around" method. That is, once the last location in the buffer is filled, the next byte is placed into the first location of the buffer. Thus, it is possible for the last byte address pointer to be pointing to an address that is less than (i.e., ahead of) the first byte address.

If any of the read buffer requests cannot be done, the 64000 returns the appropriate error code to location CA as shown in Table 8-7.

## **Updating Read/Write Buffers (8DH)**

Once the read and write buffers have been set up and opened as described in preceding paragraphs "Write to 8251" and "Read from 8251", the buffers may both be updated by using one code. To do this, the user program places the updated first and last byte address pointers for both the read and write buffers into the corresponding locations in the RS-232 I/O control buffer and then places code 8DH into location CA.

The 64000 responds to the update request as described in the "Write to 8251" and "Read from 8251" paragraphs. However, in addition to setting, monitoring, and updating the first and last byte address pointers, the number of bytes received and transmitted by the 8251 is also set, updated, and monitored. This provides an additional indication of how much data has been sent and received.

Table 8-7. RS-232 I/O Codes

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN RS-232 FILE	CA	80H	CA	00	01-07: NA
					08
					09: File already open.
					10-14: NA
CLOSE RS-232 FILE	CA	81H	CA	00	01-07: NA
					08
					09: File not open.
					10-14: NA
INITIALIZE 8251	CA	82H	CA	00	Same as 81H, above
	CA+1	Command Instruction			
	CA+2	Mode Instruction			
	CA+3	Sync Option word			
	CA+4	Sync Character, one			
	CA+5	Sync Character, two			

**Table 8-7. RS-232 I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
COMMAND TO 8251	CA	83H	CA	00	Same as 81H, above
	CA+1	Command Word			
STATUS FROM 8251	CA	84H	CA	00	Same as 81H, above
			CA+1	Status Word	
READ SINGLE BYTE FROM 8251	CA	85H	CA	00	Same as 81H, above
			CA+1	Byte Read	
WRITE SINGLE BYTE TO 8251	CA	86H	CA	00	Same as 81H, above
	CA+1	Write Byte			
OPEN WRITE BUFFER	CA	87H	CA	87H	
	CA+1	Reserved for Initialization buffer	The 64000 transfers write data from the users buffer to the 64000 buffer.		
	↓ CA+5				

**Table 8-7. RS-232 I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
	CA+6	#Bytes sent by 8251. Cleared by open (87H). Updated by 64000 when update code 89H or 8DH is used.	For each byte transferred to the 64000 buffer, first byte address pointer is incremented by one.		
	CA+7 (lsw, msb)	Buffer Begin Address			
	CA+8 (lsw, lsb)	pointer			
	CA+9 (msw, msb)				
	CA+10 (msw, lsb)				
	CA+11 (lsw, msb)	Buffer End Address			
	CA+12 (lsw, lsb)	pointer			
	CA+13 (msw, msb)				
	CA+14 (msw, lsb)				

**Table 8-7. RS-232 I/O Codes (Cont'd)**

<b>Request Name</b>	<b>User Program Request</b>		<b>64000 Response to:</b>		
	<b>Address</b>	<b>Contents</b>	<b>Valid User Request Address</b>	<b>Request Contents</b>	<b>Invalid Request Error Code</b>
	CA+15 (lsw, msb)	First Byte Address			
	CA+16 (lsw, lsb)	pointer			
	CA+17 (msw, msb)				
	CA+18 (msw, lsb)				
	CA+19 (lsw, msb)	Last Byte Address			
	CA+20 (lsw, lsb)	pointer			
	CA+21 (msw, msb)				
	CA+22 (msw, lsb)				

**Table 8-7. RS-232 I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
CLOSE WRITE BUFFER	CA	88H	CA	00	Same as 81H, above.
UPDATE WRITE BUFFER	CA	89H	CA	89H	Same as 81H, above.
	CA+1	Reserved for Initialization Buffer	The user updates the last byte address Pointer to indicate how much new write data is in the buffer. The 64000 processes the write data, increments the first byte addr. pointer, and updates # bytes sent by 8251 as required.		
	CA+5				
	CA+6	# Bytes sent by 8251.			
	CA+7 ↓ CA+14	Not changed by user.			
	CA+15 (lsw, msb)	First Byte Address pointer			
	CA+16 (lsw, lsb)				
	CA+17 (msw, msb)				
	CA+18 (msw, lsb)				
	CA+19 (lsw, msb)	Updated Last Byte Address pointer			
	CA+20 (lsw, lsb)				

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response to:		
	Address	Contents	Valid User Request Address	Invalid Request Contents Error Code	
	CA+21	(msw, msb)			
	CA+22	(msw, lsb)			

**Table 8-7. RS-232 I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response To:		
	Address	Contents	Valid User Request		Invalid Request
			Address	Contents	Error Code
OPEN READ BUFFER	CA	8AH	CA	8AH	Same as 81H, above
	CA+1	Reserved for Initialization and write buffers. # Bytes received by 8251. Cleared by open (8AH). Updated by 64000 when update code 8CH or 8DH is used.	The user sets first and last address pointers to point to buffer beginning address. The 64000 will transfer data from the 8251 to the 64000 buffer. The user must use the commands 8CH or 8DH to transfer the data to the users buffer.		
	↓				
	CA+22				
	CA+23				
	CA+24 (lsw, msb)	Buffer Begin Address pointer			
	CA+25 (lsw, lsb)				
	CA+26 (msw, msb)				
	CA+27 (msw, lsb)				
	CA+28 (lsw, msb)	Buffer End Address pointer			
CA+29 (lsw, lsb)					

**Table 8-7. RS-232 I/O Codes (Cont'd)**

Request Name	User Program Request		64000 Response to:		
	Address	Contents	Valid User Request Address	Request Contents	Invalid Request Error Code
	CA+30 (msw, msb)				
	CA+31 (msw, lsb)				
	CA+32 (lsw, msb)	First Byte Address pointer			
	CA+33 (lsw, lsb)				
	CA+34 (msw, msb)				
	CA+35 (msw, lsb)				
	CA+36 (lsw, msb)	Last Byte Address pointer			
	CA+37 (lsw, lsb)				
	CA+38 (msw, msb)				
	CA+39 (msw, lsb)				

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:			
			Valid User Request		Invalid Request	
	Address	Contents	Address	Contents	Error Code	
UPDATE WRITE/ READ BUFFERS	CA	8DH	CA	00H	Same as 81H above	
	CA+1 ↓ CA+5	Reserved for Initialization Buffer	Write and read buffers are both updated as described above.			
	CA+6 ↓ CA+22	Same as shown for update Write Buffer, above.				
	CA+23 ↓ CA+39	Same as shown for update Read Buffer, above.				

**Table 8-7. RS-232 I/O Codes (Cont'd)**

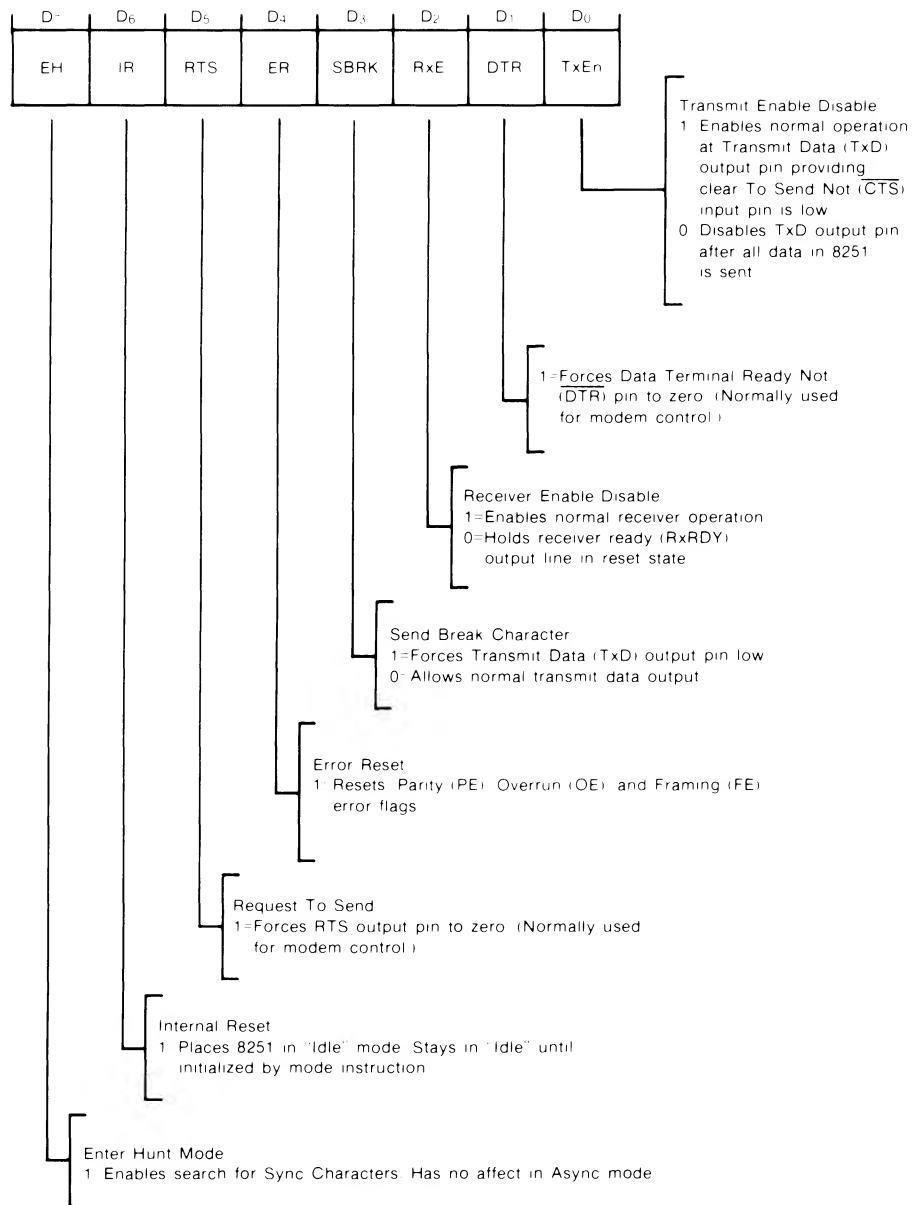
<b>Request Name</b>	<b>User Program Request</b>		<b>64000 Response to:</b>		
	<b>Address</b>	<b>Contents</b>	<b>Valid User Request Address</b>	<b>Contents</b>	<b>Invalid Request Error Code</b>
	CA+36 (lsw, msb)	Last Byte Address pointer			
	CA+37 (lsw, lsb)				
	CA+38 (msw, msb)				
	CA+39 (msw, lsb)				

**Table 8-7. RS-232 I/O Codes (Cont'd)**

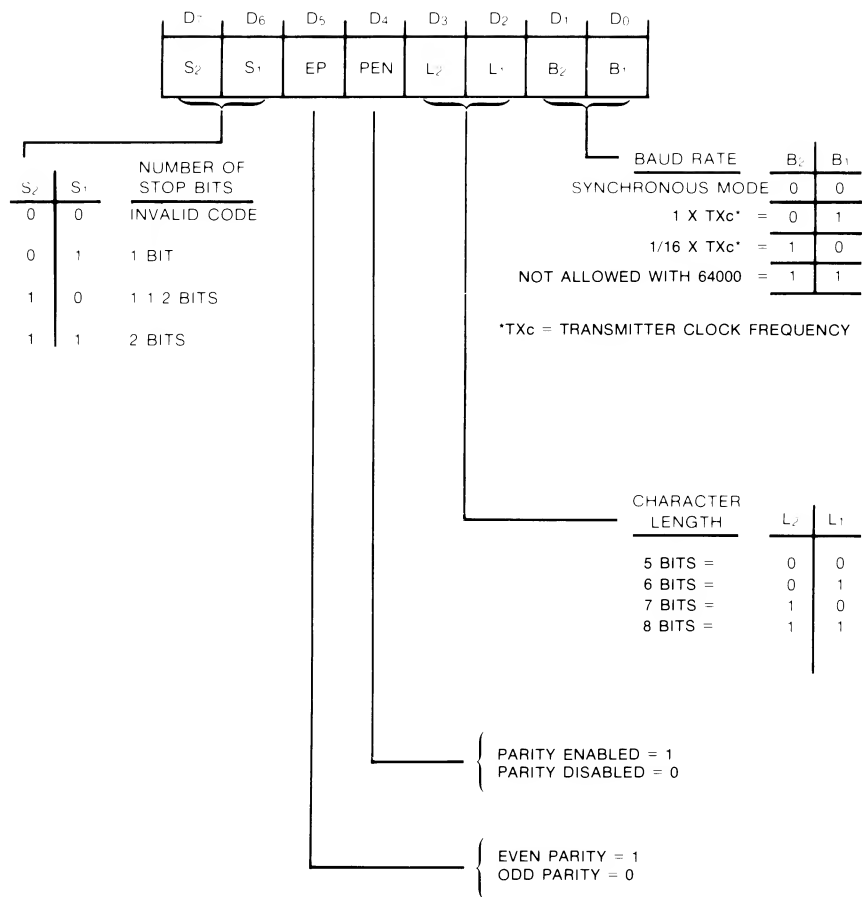
Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
UPDATE WRITE/ READ BUFFERS	CA	8DH	CA	00H	Same as 81H above
	CA+1 ↓ CA+5	Reserved for Initialization Buffer			
	CA+6 ↓ CA+22	Same as shown for update Write Buffer, above.	Write and read buffers are both updated as described above.		
	CA+23 ↓ CA+39	Same as shown for update Read Buffer, above.			

ADDRESS	ASYNCHRONOUS MODE - INITIALIZATION FORMAT	SYNCHRONOUS MODE- SINGLE SYNC CHARACTER INITIALIZATION FORMAT	SYNCHRONOUS MODE - DOUBLE SYNC CHARACTER INITIALIZATION FORMAT	ADDRESS
CA	82H - INITIALIZE 8251	82H - INITIALIZE 8251	82H - INITIALIZE 8251	CA
CA+1	COMMAND INSTRUCTION (Internal Reset 8251)	COMMAND INSTRUCTION (Internal Reset 8251)	COMMAND INSTRUCTION (Internal Reset 8251)	CA+1
CA+2	ASYNCHRONOUS MODE INSTRUCTION	SYNCHRONOUS MODE INSTRUCTION	SYNCHRONOUS MODE INSTRUCTION	CA+2
CA+3	SYNC OPTION WORD 0=No sync characters	SYNC OPTION WORD 1=1 sync character	SYNC OPTION WORD 2=2 sync characters	CA+3
CA+4	Not Used	SYNC CHARACTER 1	SYNC CHARACTER 1	CA+4
CA+5	Not Used	Not Used	SYNC CHARACTER 2	CA+5
CA+6 ↓ CA+22	RESERVED FOR WRITE CONTROL	RESERVED FOR WRITE CONTROL	RESERVED FOR WRITE CONTROL	CA+6 ↓ CA+22
CA+23 ↓ CA+39	RESERVED FOR READ CONTROL	RESERVED FOR READ CONTROL	RESERVED FOR READ CONTROL	CA+23 ↓ CA+39

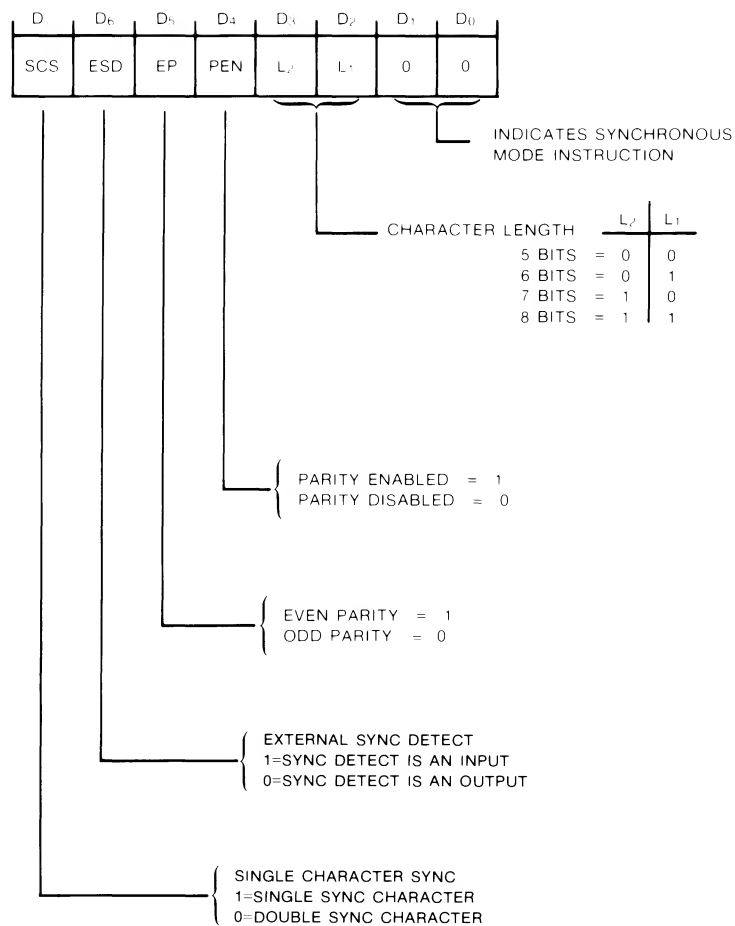
**Figure 8-8. 8251 Initialization Formats**



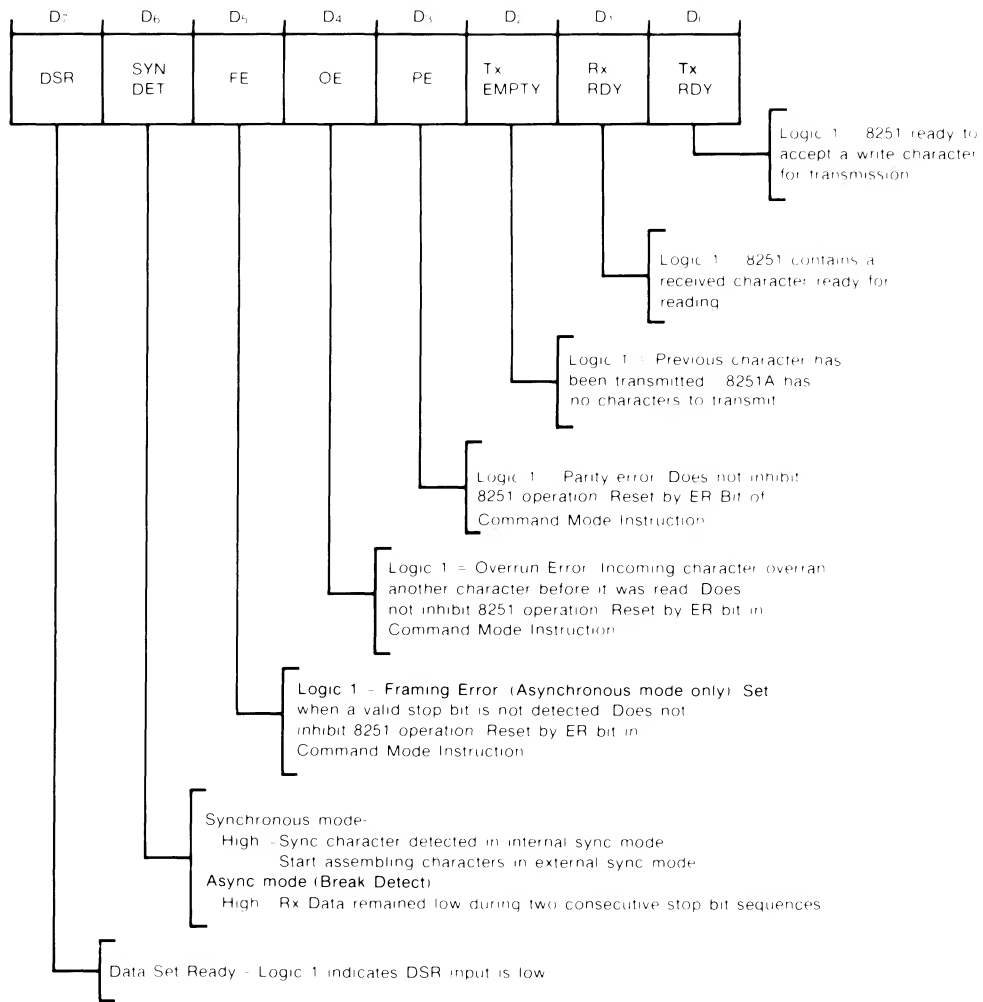
**Figure 8-9. Command Mode Instruction Format**



**Figure 8-10. Asynchronous Mode Instruction Format**



**Figure 8-11. Synchronous Mode Instruction Format**



**Figure 8-12. 8251 Status Word Format**

Phase I - User Sets Up Write Buffer

User sets up write buffer as follows:

1. Assigns buffer beginning and ending addresses: WBUFEBEG and WBUFEND.
2. Writes block of characters into buffer shown as first byte through last byte.

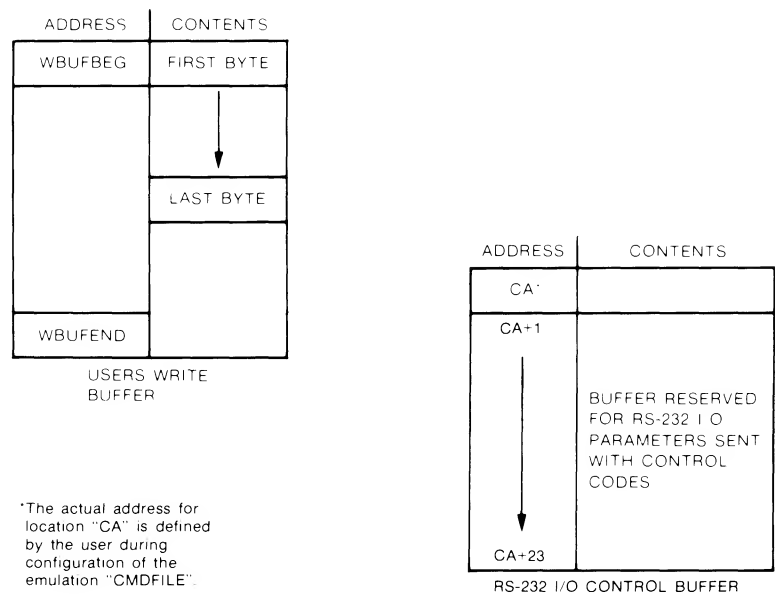
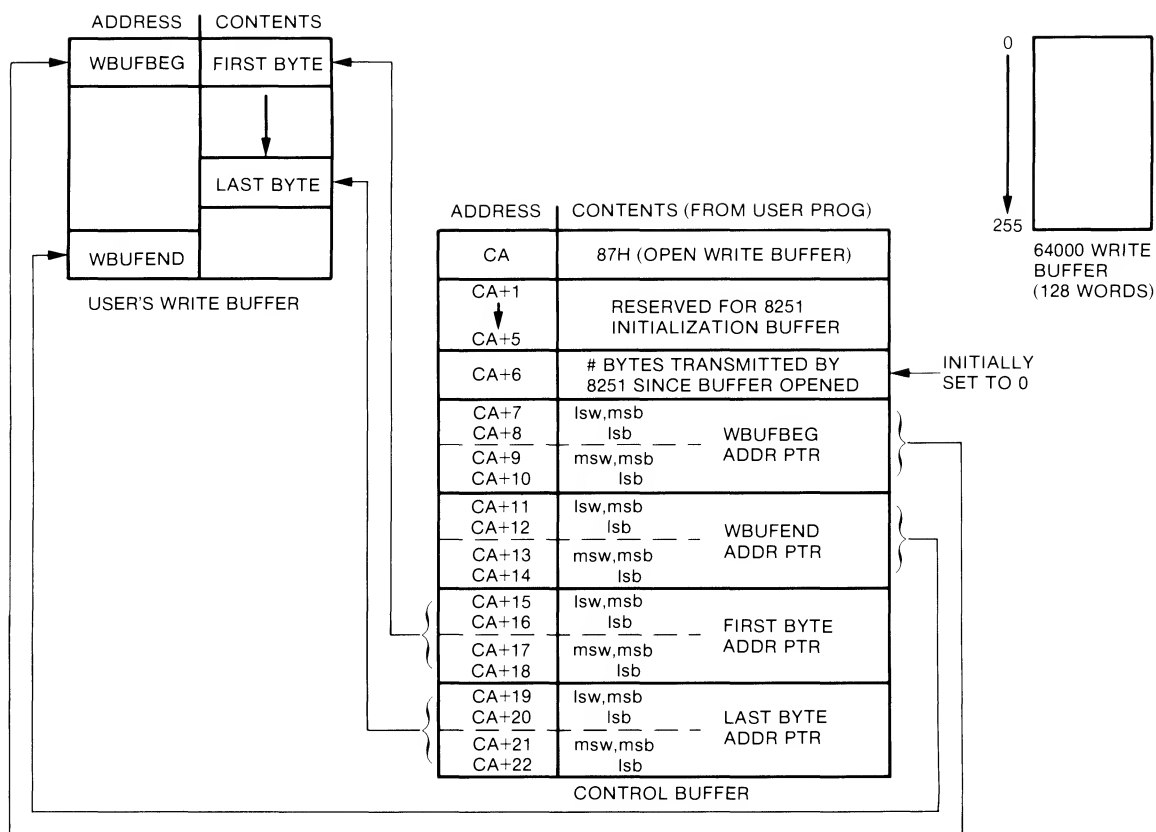
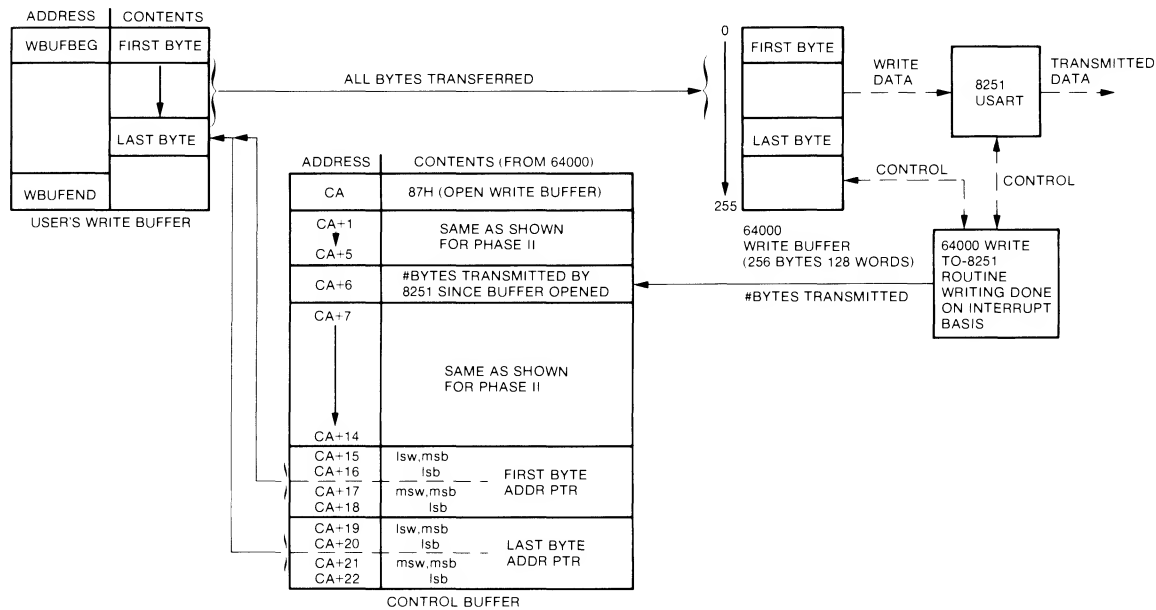


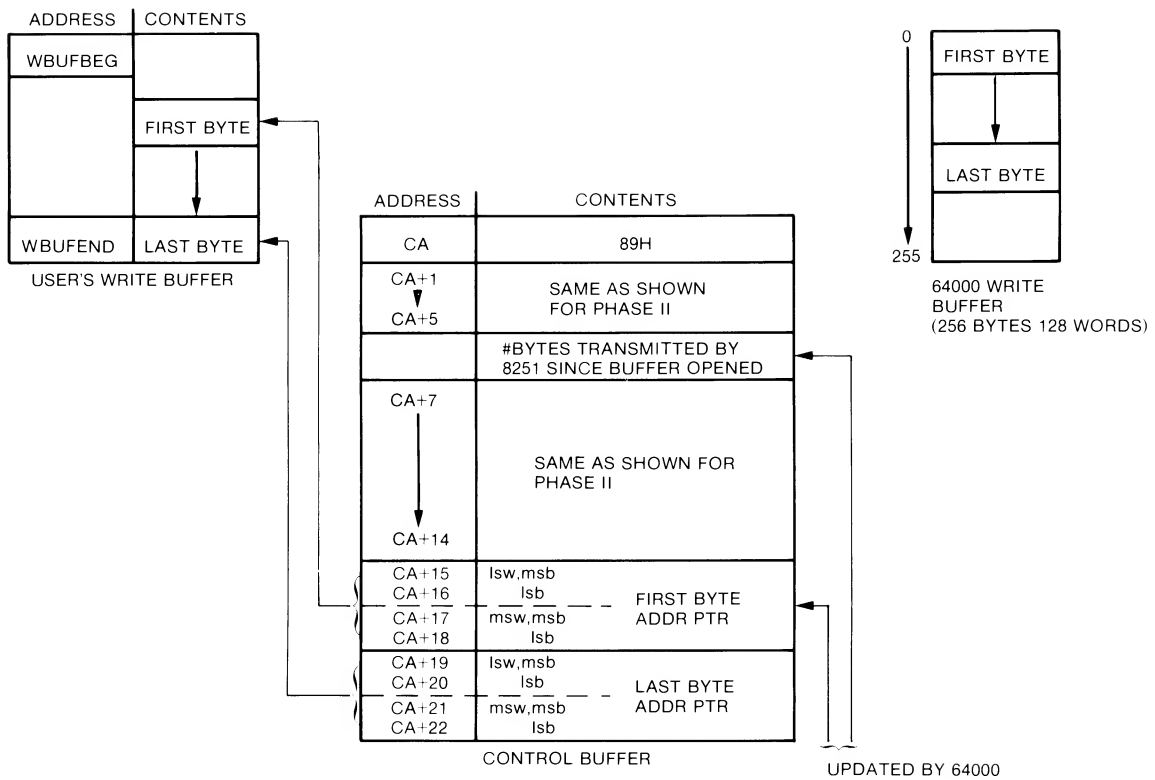
Figure 8-13. Writing RS-232 Record - Phase I



**Figure 8-14. Writing RS-232 Record - Phase II**



**Figure 8-15. Writing RS-232 Record - Phase III**



**Figure 8-16. Writing RS-232 Record - Phase IV**



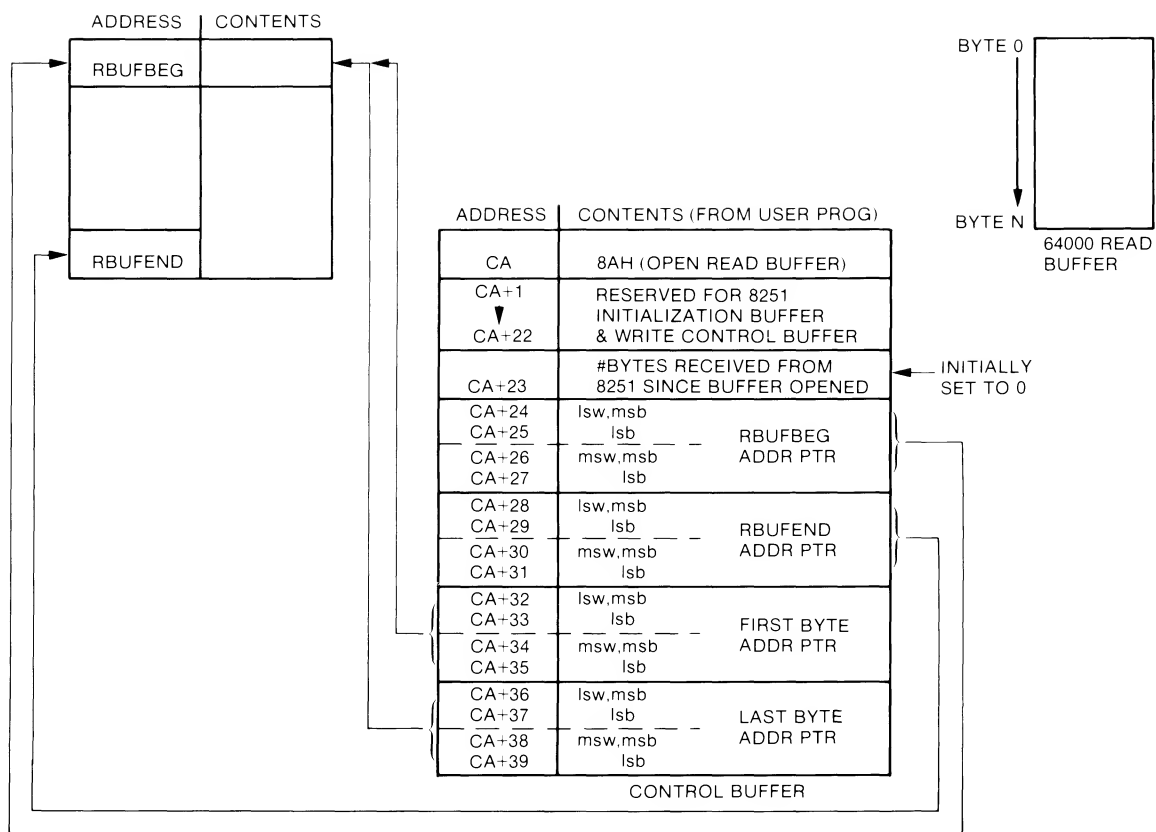
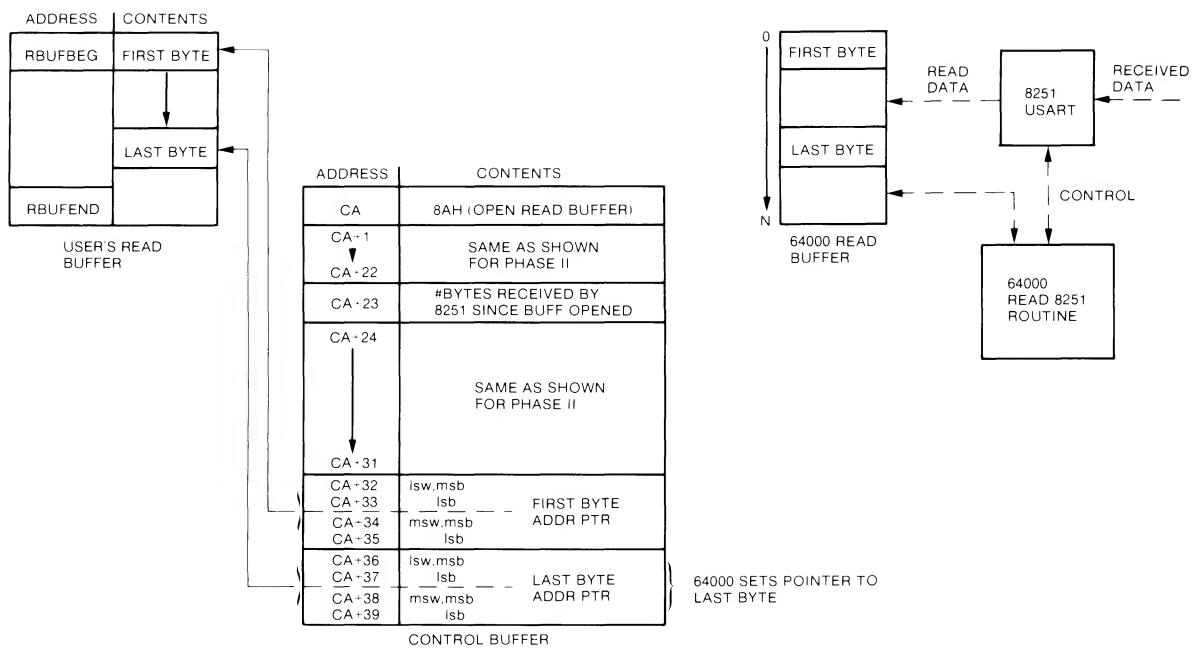
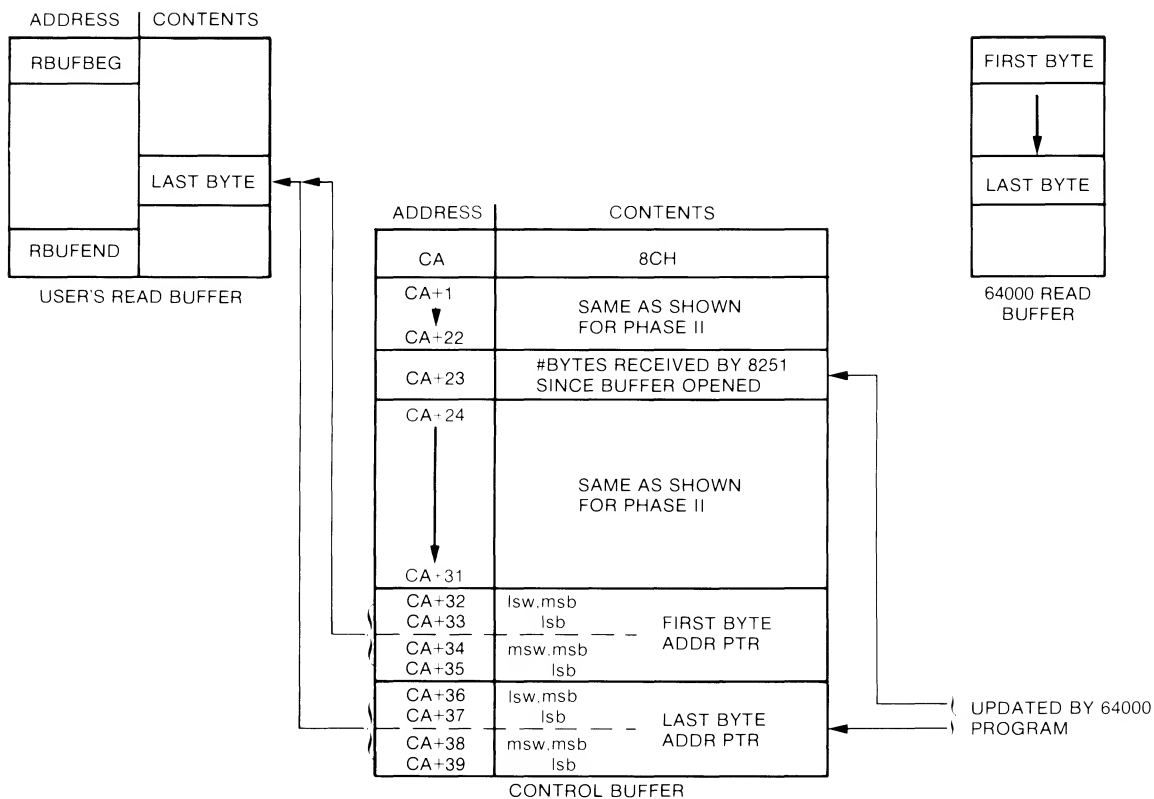


Figure 8-18. Reading RS-232 Record - Phase II



**Figure 8-19. Reading RS-232 Record - Phase III**



**Figure 8-20. Reading RS-232 Record - Phase IV**

## Simulated I/O Error Codes

The general definitions for the simulated I/O error codes are listed in Table 8-8. Where applicable, more specialized definitions of these error codes are listed in individual I/O code Tables, 8-1, 8-2, etc.

When a request by the user program cannot be executed, the applicable error code is returned by the 64000 program to location CA.

**Table 8-8. Simulated I/O Error Codes - General Definitions**

<b>Decimal Code #</b>	<b>(Hex)</b>	<b>Meaning</b>
00		No error - successful operation
01		End of file
02		Invalid disc
03		File not found
04		File already exists
05		No disc space available
06		No directory space available
07		File is Corrupt (bad linkage)
08		Cannot read/write assigned memory
09		Request not allowed
10	( A )	Invalid file type
11	( B )	Invalid row or column no.
12	( C )	Invalid record length
13	( D )	Invalid display character >OFOH
14	( E )	While in simulated display I/O or simulated keyboard I/O, the 64000 "simulate" soft key was pressed to exit simulate I/O. All open files are closed.
15	( F )	Error in new disc file name when attempting to change a disc file name. First character in file name limited to capital letters A through Z. Second and following characters may contain capital and lower case letters, numerals 0 through 9, underlines, and only if required to fill in the last byte of the last word, a blank is used.

# Simulated I/O Sample Programs

The following Figures show the listing for source programs which actually use simulated I/O facilities. The programs are real and do work.

Figure	Sample Program Type
8-21	Simulated Display I/O - Sample Program A
8-22	Simulated Display I/O - Sample Program B
8-23	Simulated Keyboard, Display, and One Disc File I/O - Sample Program
8-24	Simulated Keyboard, Display, and Two Disc Files I/O - Sample Program

“8080”

\*This 8080 program uses the simulated display I/O interface.

\*The display is opened, and two messages are written; one to

\*row/column 2,40 and one to row/column 18,20.

\*Control address for display is 0000H.

\*Program execution should start at 0200H.

```
                ORG 200H
USR_ADR EQU 0
START          LXI SP,100H
*OPEN          DISPLAY
                MVI A,80H
                STA USR_ADR
                CALL WAIT
*SET           ROW/COLUMN 2/40
                MVI A,2
                STA USR_ADR+1
                MVI A,40
                STA USR_ADR+2
                MVI A,83H
                STA USR_ADR
                CALL WAIT
```

**Figure 8-21. Simulated Display I/O - Sample Program A**

```

*WRITE MESSAGE 1
    LXI H,MESSAGE_1
    CALL XFR_MSG
    CALL WAIT
*SET ROW/COLUMN 18/20
    MVI A,18
    STA USR_ADR+1
    MVI A,20
    STA USR_ADR+2
    MVI A,83H
    STA USR_ADR
    CALL WAIT
*WRITE MESSAGE 2
    LXI H,MESSAGE_2
    CALL XFR_MSG
    CALL WAIT
*LOOP HERE, LEAVE DISPLAY OPEN
    JMP $

*WAIT FOR USR_ADR=0 ... IO REQUEST COMPLETED
WAIT
    LDA USR_ADR
    CPI 0
    JM WAIT
    RET

*TRANSFER MESSAGE FROM C(DE) TO USR_ADR+1
*C(DE(0))=#BYTES
*    C(DE(1))=BYTE 1
*
*    .
*
*    .
*
*    .
*    C(DE(N))=BYTE N
*USR_ADR=84H ... REQUEST TO WRITE ROW/COLUMN
XFR_MSG
    XCHG
    LDAX D
    MOV B,A                ;B=#BYTES
    INX D
    LXI H,USR_ADR+1        ;HL=USR_ADR+1
    INX H
    MOV M,A
    INX H

```

**Figure 8-21. Simulated Display I/O - Sample Program A (Cont'd)**

XFR_LOOP	LDAX D	;GET DATA BYTE
	MOV M,A	;AND STORE IN USR_ADR BUFFER
	INX D	
	INX H	
	DCR B	;AND CHECK FOR COMPLETION
	MOV A,B	
	CPI 0	
	JNZ XFR_LOOP	
	MVI A,84H	;SET REQ FOR WRITE ROW/COLUMN
	STA USR_ADR	
	RET	
*MESSAGE 1		
MESSAGE_1	DB 12	;#BYTES
	ASC "Display test"	
*MESSAGE 2		
MESSAGE_2	DB 8	;#BYTES
	ASC "End test"	
	END START	

**Figure 8-21. Simulated Display I/O - Sample Program A (Cont'd)**

This program scrolls ASCII characters onto the 64000 Display.

```

"8080"
DISP      EQU      0D00H          ;CONTROL ADDRESS FOR
                                   ; SIMULATED DISPLAY IO
START     LXI      SP,1000H      ;STACK DOWN FROM 0FFFH
          LXI      H,DISP        ;LOAD H WITH CA
          CALL     CHECK         ;WAIT FOR SERVICE
          MVI      B,0           ;INITIALIZE COUNTER
LOOP      INX      H             ;SET M TO DISP+1
          MVI      M,4           ;SET BYTE COUNT TO FOUR
                                   ; CHARACTERS/LINE
          INX      H             ;SET M TO DISP+2
                                   ;THEN EACH LINE IS:
          MVI      M,32          ;ASCII BLANK (1ST CHAR)
          INX      H
          INR      B
                                   ;THEN:
          MOV      M,B           ;2ND CHAR
          INX      H            ;THEN
          MVI      M,32          ;ANOTHER BLANK (3RD CHAR)
          INX      H
          MVI      M,0           ;AND A "NULL" (4TH CHAR)
          LXI      H,DISP
          MVI      M,82H         ;REQ "SCROLL" 1 LINE
          CALL     CHECK         ;WAIT FOR SERVICE
          MVI      A,127         ;AVOID SPECIAL CHARACTERS
          CMP      B             ;IF B > 127 RESET B TO 0
          JNZ      LOOP
          MVI      B,0
          JMP      LOOP
CHECK     LDA      DISP
          CPI      0             ;WAIT UNTIL CA=0
          RZ
          JMP      CHECK
          END      START

```

**Figure 8-22. Simulated Display I/O - Sample Program B**

```

"8080"
    ORG 0
* THIS PROGRAM OPENS KEYBOARD AND DISPLAY FILE. THEN, UPON
* CARRIAGE RETURN IT COPIES KEYBOARD DATA TO DISPLAY AND FILE
* FIO.

* OPEN DISPLAY AND KEYBOARD
START
    LXI SP,400H
    MVI A,80H          ;OPEN DISPLAY
    STA DSP_BUF
    STA KEY_BUF
    MVI A,02H          ;DELETE FILE
    STA FILE_BUF+1     ;TYPE 2(SCR)
    MVI A,0            ;DISC#0
    STA FILE_BUF+2
    MVI A,83H
    STA FILE_BUF
WAIT_FILE_D            ;WAIT FR FILE DELETE
    LDA FILE_BUF
    CPI 0
    JM WAIT_FILE_D
    MVI A,80H          ;CREATE FILE
    STA FILE_BUF
WAIT_FILE_C            ;WAIT FOR FILE CREATE
    LDA FILE_BUF
    CPI 0
    JM WAIT_FILE_C
NEXT_KEY_DATA
    MVI A,-2           ;NOW SETUP KEYBOARD FOR CMD=-2
    STA KEY_BUF+1
    MVI A,240          ;AND MAX # CHARS
    STA KEY_BUF+2
    LDA K_CMD          ;AND OPEN/READ KEYBOARD
    STA KEY_BUF
* WAIT FOR CR(CMD>=0)
WAIT_FOR_CR
    LDA KEY_BUF
    CPI 0

```

**Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O  
Sample Program**

```

        JM WAIT_FOR_CR
        LXI D,DSP_BUF+1
        CALL XFR_DATA
* WRITE TO DISPLAY
        MVI A,82H
        STA DSP_BUF
WAIT_FOR_DSP
        LDA DSP_BUF
        CPI 0
        JM WAIT_FOR_DSP
* WRITE TO FILE FIO
        LXI D,FILE_BUF+1
        CALL XFR_DATA
        LDA KEY_BUF+3
        INR A
        STC
        CMC
        RAR
        STA FILE2BUF+1      ;SET # WORDS
        MVI A,89H
        STA FILE_BUF
WAIT_FILE_W      ;WAIT FOR FILE WRITE
        LDA FILE_BUF
        CPI 0
        JM WAIT_FILE_W
        JMP NEXT_KEY_DATA
*
* TRANSFER KEY BOARD DATA TO DISPLAY
XFR_DATA
        LXI H,KEY_BUF+3
        MOV B,M      ;GET # BYTES
        MOV A,B
        STAX D
        INX D
        INX H

```

**Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O  
Sample Program (Cont'd)**

```

XFR_LOOP
    MOV A,M
    STAX D
    INX D
    INX H
    DCR B
    JNZ XFR_LOOP
    MVI A,0
    STAX D
    RET
K_CMD DB 80H
* DISPLAY BUFFER
DSP_BUF EQU 100H
* KEYBOARD BUFFER
KEY_BUF EQU 200H
FILE_BUF EQU 300H
    ORG 100H
    DB 0
    ORG 200H
    DB 0
    ORG 300H
    DB 0
END START

```

**Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O  
Sample Program (Cont'd)**

"8080"

ORG 0  
\* THIS PROGRAM OPENS KEYBOARD, DISPLAY AND 2 FILES. THEN UPON  
\* CARRIAGE RETURN IT COPIES KEYBOARD DATA TO DISPLAY AND TO  
\* FILES F1 AND F2.

\* OPEN DISPLAY AND KEYBOARD

START

```
    LXI SP,400H      ;STACK 03FFH AND BELOW
    MVI A,80H        ;OPEN DISPLAY
    STA DSP_BUF
    STA K_CMD
    MVI A,2
    STA FB1+1        ;TYPE 2(SOURCE)
    MVI A,0           ;DISC#0
    STA FB1+2        ;BOTH FILES ON DISC 0
    STA FB2+2
    MVI A,83H
    STA FB1
WAIT_FILE_D1          ;WAIT FOR FILE DELETE
    LDA FB1
    CPI 0
    JM WAIT_FILE_D1
    MVI A,80H        ;CREATE FILE
    STA FB1
WAIT_FILE_C1          ;WAIT FOR FILE CREATE
    LDA FB1
    CPI 0
    JM WAIT_FILE_C1
    MVI A,2
    STA FB2+1        ;TYPE 2(SCR)
    MVI A,83H
    STA FB2
WAIT_FILE_D2          ;WAIT FOR FILE DELETE
    LDA FB2
    CPI 0
    JM WAIT_FILE_D2
```

**Figure 8-24. Simulated Keyboard, Display, and Two Disc Files  
I/O Sample Program**

```

        MVI A,80H      ;CREATE FILE
        STA FB2
WAIT_FILE_C2      ;WAIT FOR FILE CREATE
        LDA FB2
        CPI 0
        JM WAIT_FILE_C2
NEXT_KEY_DATA
        MVI A,-2      ;NOW SETUP KEYBOARD FOR CMD=-2
        STA KEY_BUF+1
        MVI A,240      ;AND MAX # CHARS
        STA KEY_BUF+2
        LDA K_CMD      ;AND OPEN/READ KEYBOARD
        STA KEY_BUF
* WAIT FOR CR(CMD>=0)
WAIT_FOR_CR
        LDA KEY_BUF
        CPI 0
        JM WAIT_FOR_CR
        LXI D,DSP_BUF+1
        CALL XFR_DATA
* WRITE TO DISPLAY
        MVI A,82H
        STA DSP_BUF
WAIT_FOR_DSP
        LDA DSP_BUF
        CPI 0
        JM WAIT_FOR_DSP
* WRITE TO FILE F1
        LXI D,FB1+1
        CALL XFR_DATA
        LDA KEY_BUF+3
        INR A
        STC
        CMC
        RAR
        STA FB1+1      ;SET # WORDS
        MVI A,89H
        STA FB1

```

**Figure 8-24. Simulated Keyboard, Display, and Two Disc Files  
I/O Sample Program (Cont'd)**

```

WAIT_FILE_W1          ;WAIT FOR FILE WRITE
    LDA FB1
    CPI 0
    JM WAIT_FILE_W1
* WRITE TO FILE F2
    LXI D,FB2+1
    CALL XFR_DATA
    LDA KEY_BUF+3
    INR A
    STC
    CMC
    RAR
    STA FB2+1          ;SET # WORDS
    MVI A,89H
    STA FB2
WAIT_FILE_W2          ;WAIT FOR FILE WRITE
    LDA FB2
    CPI 0
    JM WAIT_FILE_W2

    JMP NEXT_KEY_DATA
*
* TRANSFER KEY BOARD DATA TO DISPLAY
XFR_DATA
    LXI H,KEY_BUF+3
    MOV B,M            ;GET # BYTES
    MOV A,B
    STAX D
    INX D
    INX H
XFR_LOOP
    MOV A,M
    STAX D
    INX D
    INX H
    DCR B

```

**Figure 8-24. Simulated Keyboard, Display, and Two Disc Files  
I/O Sample Program (Cont'd)**

```

                JNZ XFR_LOOP
                MVI A,0
                STAX D
                RET
K_CMD          DB 80H
                ORG 100H    ;CONTROL ADDRESS FOR DISPLAY
DSP_BUF        DB 0

                ORG 200H    ;CONTROL ADDRESS FOR KEYBOARD
KEY_BUF        DB 0

                ORG 300H    ;CONTROL ADDRESS FOR FILE 1
FB1            DB 0

                ORG 400H    ;CONTROL ADDRESS FOR FILE 2
FB2            DB 0
                END START

```

**Figure 8-24. Simulated Keyboard, Display, and Two Disc Files  
I/O Sample Program (Cont'd).**

## 64000 File Formats

The 64000 file accessible to the user through the simulated disc file I/O interface are described in the following paragraphs.

### Assembler Symbols File (File Type 12)

This file contains the symbols and their corresponding values assigned by the assembler. It also indicates the symbol type. Symbols may be either ABS (absolute), or relocatable to the PROG, DATA, or COMN areas. (These terms are all defined in the 64000 Assembler/Linker Reference Manual.)

The assembler symbols file is generated each time a source program containing symbols is assembled into an object file. The file consists of a group of records with each record in turn consisting of up to 128 sixteen-bit words (0-127). Each record must be structured as follows: (See Figures 8-25 and 26.)

- o Record Identification (ID) Word
- o Symbol Definition Blocks (Length variable from two to ten words.)
- o Checksum Word

Each of the three items is described in the following paragraphs.

#### **Record ID word**

The ID word is always the first word in each record and contains the number “6”. (The “6” is used internally and is not to be confused with the file type number which is 12.)

#### **Symbol definition blocks**

A symbol definition block consists of the symbol word(s) and the value word(s). (See Figure 8-27.)

Symbol word(s) - The ASCII character, or characters, are contained in this word (or words). From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated to 15 characters.)

First symbol word - The first word in each symbol definition block is structured the same. The least significant eight bits (7 thru 0) contain the first ASCII character in the symbol. The most significant eight bits (15 thru 8) always contain the following information:

- o Symbol Length (SL) - Bits 15, 14, and 13 specify the number of symbol words -1 in this block. (See Figure 8-28, Example A.) For example, if the symbol consists of two ASCII characters, which require two symbol words, SL is equal to 1. Examples of symbols made up of one to five characters, which require one and three words respectively, are shown in Figure 8-28, examples B and C.
- o Reserved Bits - Bits 12, 11, and 10 contain 000 and are reserved for use by other program modules.
- o Memory Relocation (Relo) - Bits 9 and 8 specify how the symbol may be relocated as follows:

Bit 9	Bit 8	Storage Type
0	0	ABS (Absolute)
0	1	PROG area
1	0	DATA area
1	1	COMN area

Additional symbol words - The second thru the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (Code 20H). That is, the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words are required to define a symbol, then only five symbol words must be used.

Value word(s) - Immediately following the last symbol word may be either one or two value words, depending upon the size of the target processors addressable memory. This word, or words, specifies the value assigned to the symbol by the assembler. If the value can be contained in one 16-bit word, then only one word is to be used. Two 16-bit words are used only if they are both required. When two words are used, the first word contains the least significant 16-bits and the second word contains the most significant 16 bits.

All symbol definition blocks within the assembler symbol file must be structured as defined above.

#### **Checksum word**

The checksum word must be the last word in the assembler symbols file. If the file is completely full, then the checksum word will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the file.

### **User Buffer/Assembler Symbols File Packing Formats**

The format relationship between the user buffer when reading from, or writing into, a 64000 Assembler Symbols File is shown in Figure 8-27.

#### **Linker Symbol File (File Type 13)**

The Linker Symbol File is generated anytime program modules are linked together. It consists of the following four types of records (see figure 8-29):

- **TYPE 1 RECORD** - Microprocessor Configuration Record (one per file).
- **TYPE 2 RECORD** - Global Symbols Records.
- **TYPE 3 RECORD** - Program Names Records.
- **TYPE 4 RECORD** - Memory Space Allocation (RANGE).

Each of the above listed records is described below.

**TYPE 1 RECORD** (see figure 8-30). The first record in the Linker Symbol File is always a TYPE 1 record. It is similar to the NAME record in relocatable files and is required for the linker to configure itself for the correct microprocessor. The record is only used when a link\_sym file is the first file given as a response to the linker question "Object files?". This is a fixed-length record containing 26 words and is configured as follows:

- a. Record identification (ID) word - The record ID word is always the first word in the record. It is also the first word in the Linker Symbol File and contains the number "1". This number identifies the record as the microprocessor configuration record. (The "1" is used internally and should not be confused with the file type number which is "13").
- b. Pad words 1 through 15 - These words are inserted so that word positions 16 through 23 in this name record contains the same information as do corresponding word positions in the name records of the relocatable files.
- c. Name and user ID word block - A fixed length 8-word block (words 16 through 23) that contains the microprocessor configuration file name in standard file name format, i.e., I68000:HP. The MSB of word 16 contains the following information:

bits 15-13: indicates the number of 16-bit words-1 in the file name.

bits 12-11: indicates the number of 16-bit words in the userid.

bits 10-8 : "don't care" conditions.

- d. Address size - This word (word 24) is required for emulation and state analysis. It defines the number of 16-bit words required to specify an address for the target processor. The LSB of this word indicates the address size (1 = one word addresses (16 bits); 2 = two-word addresses (32 bits)). The MSB of this word, hishift (see figure 8-30), is used to convert 32-bit logical addresses (segment, offset) to physical addresses. This is accomplished by putting the segment in the MS 16 bits of a 32-bit register, shift right the number of bits indicated in hishift, then do a 32-bit add to offset.
- e. Checksum - The checksum word (word 25) contains the arithmetic sum of the binary values of the preceding 25 words in this record.

**TYPE 2 RECORD** (see figure 8-31). The Linker Symbol File may contain multiple Global Symbol Records (TYPE 2). The first Global Symbol Record follows the Microprocessor Configuration Record and all subsequent Global Symbol Records are contiguous. These records are copied from the linker's symbol table at the conclusion of pass 1.

A Global Symbols Record contains the global symbols and the relocated address values (symbol values) generated when the program modules are linked. Each record may consist of up to 128 16-bit words (0-127 words) structured as follows (see figure 8-32):

- a. Record identification (ID) word - The record ID is always the first word in each record and contains the number "2". (The "2" is used internally and is not to be confused with the file type number which is "13".)
- b. Global symbol definition blocks - A global symbol definition block consists of the symbol word(s) and the value word(s) which are described in more detail in this paragraph.
- c. Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum word will be the 128th word (word #127).

Symbol word(s) - The ASCII character, or characters, are contained in this word (or words). From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated as 15 characters.)

First symbol word - The first word in every symbol definition block is structured the same. The least significant eight bits (7 through 0) contain the first ASCII character in the symbol. The most significant eight bits (15 through 8) always contains the following information (see figure 8-32):

- a. Global Symbol Length (GSL) - Bits 15, 14, and 13 specify the number of symbol words-1 in this block. For example, if the global symbol consists of two ASCII characters, which require two symbol words, GSL is equal to 1. (The second byte in the second word will contain an ASCII blank, i.e., code 20H.)
- b. Bits 12, 11, and 10 - "don't care" conditions.
- c. Memory Relocation (Relo) - Bits 9 and 8 specify how the symbol may be relocated as follows:

Bit 9	Bit 8	Storage Type
0	0	ABS (Absolute)
0	1	PROG area
1	0	DATA area
1	1	COMN area

Additional symbol words - The second through the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (code 20H). That is the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words required to define a symbol, then only five symbol words must be used.

Symbol value word(s) - Immediately following the last symbol word may be either one or two value words, depending upon the size of the target processor addressable memory. This word (or words) specifies the address assigned to the symbol by either the assembler (if ABS-absolute) or by the linker. If the address can be contained in one 16-bit word, then only one word is to be used. Two 16-bit words are used only if they are both required. When two words are used, the first word contains the least significant 16 bits and the second word contains the most significant bits of the symbol address.

All global symbol definition blocks within the Linker Symbol File must be structured as defined above.

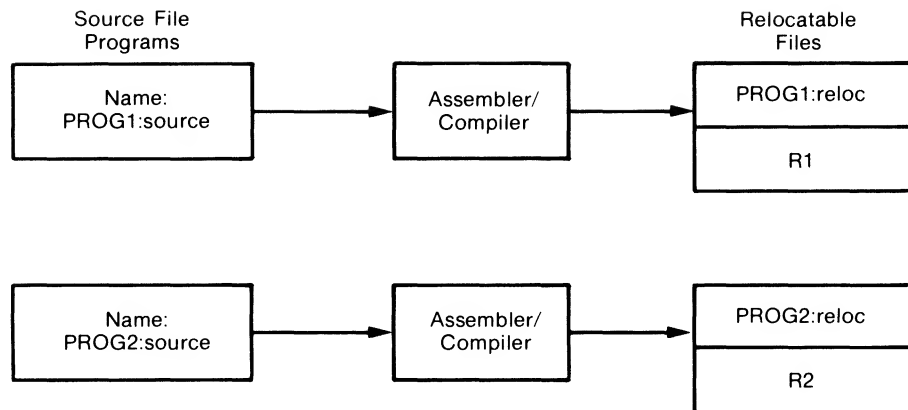
**TYPE 3 RECORDS** (see figure 8-33). The Linker Symbol File may contain multiple Program Names Records. The first Program Names Record follows the last Global Symbols Record. All succeeding Program Names Records are contiguous.

The names of type 3 records are not maintained in any internal structure. Program names have an implicit ordinal number value from 0 to N. It should be noted that if a link\_sym file is given as an input to the linker, the resulting link\_sym file does not contain the program names from the input link\_sym file.

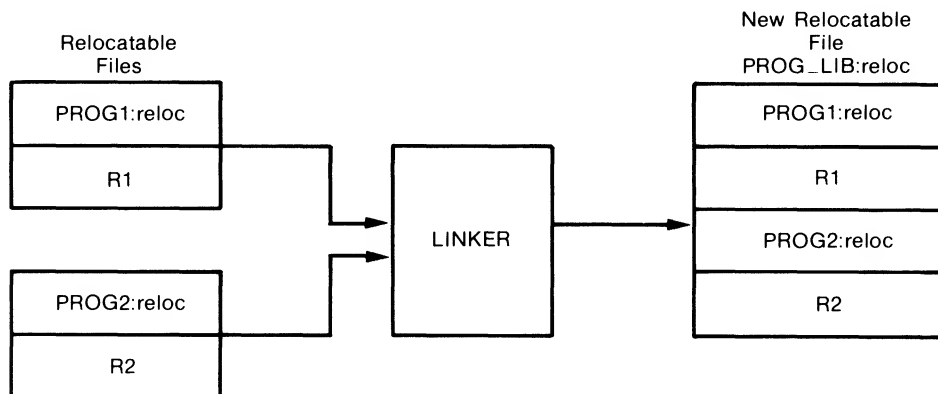
Type 3 records contain all source program names and their relocation addresses. The primary purpose of these records is to provide relocation addresses for the symbols in asm\_sym files.

Program names are not the same as file names. The most common example of this is with libraries. Program names come from the Program Description Records within Relocatable Files (File Type 3 - see figure 8-39). The name in the relocatable record (see figure 8-40) is the name of the source file that produced the relocatable file. The program name will be the same as the relocatable file name as long as the relocatable file has not been renamed or copied to a library.

For example, if two separate source file programs are assembled/compiled, the result will be two separate relocatable files with each having the file name of the source program as follows:



If the two relocatable files are linked together to form a library, for example, a new relocatable file would be built under a new file name as follows:



The linker output listing for the above would be:

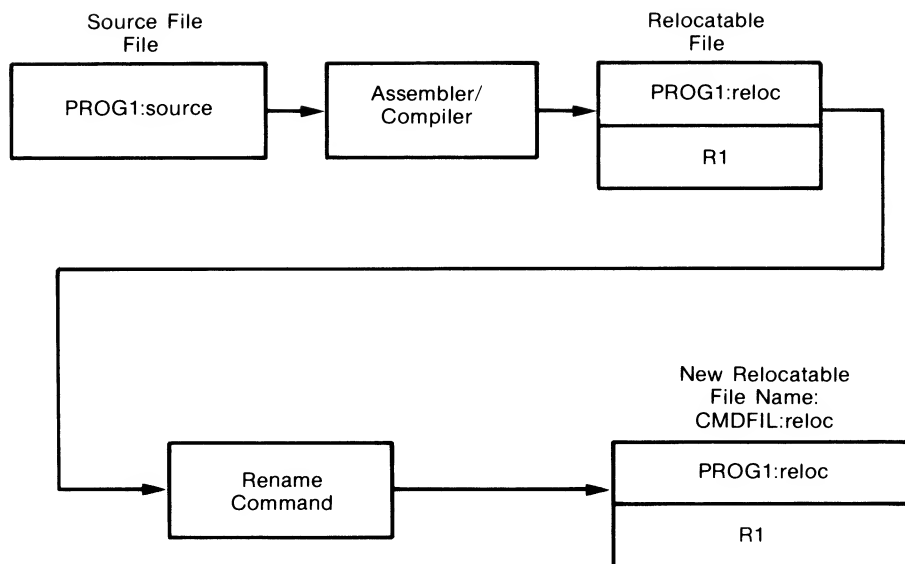
```

PROG_LIB:ID
PROG1:ID
PROG2:ID

```

Note the two original source file names are indented, indicating multiple relocatables in PROG\_LIB:ID file.

Using the 64000 system “rename” command will also result in a relocatable file having a different name than the source file program as follows:



The linker output listing for the above would be:

```

CMDFIL:ID
PROG1:ID
  
```

A Program Names Record contains the names of the source file programs, the corresponding user ID's and the load addresses generated when the program modules are linked. Each record may consist of up to 128 sixteen-bit words (words 0-127) structured as follows:

- a. One Record Identification (ID) Word.
- b. Multiple Program Name and Addresses Definition Blocks (fixed length blocks of 14 words each).
- c. One checksum word.

Record identification (ID) word - The ID word is always the first word in each record and contains the number "3". (The "3" is used internally and is not to be confused with the file type number which is "13".)

Program name and addresses definition block - This is a fixed length block consisting of 14 sixteen-bit words allocated as follows (see figure 8-34):

- a. Eight words reserved for the program name and users ID.
- b. Six words reserved for the linker load addresses.

Program name and user ID words - The formatting and packing of these words are done in the same way as described for the Microprocessor Configuration Record (TYPE 1), Name and ID word Block.

Load address words - These words contain the load addresses assigned by the linker. If an address is not assigned to a particular area, the address words contain zeros (0000H). The MS 16-bit address word will be used only if required by the target microprocessor's addressable memory space.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, the the checksum word will be in the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

**TYPE 4 RECORDS** (see figure 8-34A). Type 4 records follow type 3 records and contain a list of memory spaces used by the relocatable files. Each block contains file, program name, and relocation information plus the lower and upper bounds of the piece of memory used. Blocks are sorted on lower bound from smallest to largest.

Records contain from 1 to 9 fixed length blocks with each block containing 14 words. A block may not cross a record boundary.

## **User Buffer/Linker Symbols File Packing Formats**

The format relationship between the user buffer when reading or writing into a 64000 Linker Symbols File is the same as shown for the Assembler Symbols File in Figure 8-27.

### **Source File (File Type 2)**

The source file is generated by the programmer from the applicable microprocessor opcodes and assembler pseudo instructions. It consists of a series of ASCII records. (See Figures 8-35 and 8-36.)

Each ASCII source record in the file is structured the same. An ASCII source record is of variable length and may contain up to 128 sixteen-bit words. Each 16-bit word contains two 8-bit ASCII bytes. If the last byte in the last word of a record is not used, it must contain an ASCII blank (20H).

The format relationship between the user buffer when reading from or writing into a 64000 source file is also shown in Figure 8-36.

### **Listing File (File Type 5)**

The listing file is a copy of a source file. It may be produced when listing to a printer, a display, etc. The format is identical to that described above, and shown in Figures 8-35 and 8-36 for the source file.

### **Absolute File (File Type 4)**

Absolute file is generated when the linker produces an absolute image of an object file or files. The absolute file contains two types of records; the first record and the additional records which follow the first record. (See Figures 8-37 and 8-38.)

**First record**

The first record has a fixed length of four 16-bit words. The first word (word 0) specifies the processor's data bus width (8, 16, etc.). The second word (word 1) specifies the data width base of the target microprocessor. The data width base is the minimum addressable entity (i.e. group of bits) used by the microprocessor. Normally this will be eight bits, but not always.

The last two words specify the transfer address value loaded into the target microprocessor's program counter. The most significant transfer address word (bits 31 thru 16) is used only if required. If not used it will contain 0000H.

**Additional records**

All records following record one are formatted the same. Each is a variable length record consisting of up to 128 sixteen-bit words (0-127).

The first word in the record (word 0) specifies the number of data bytes in the record (2 bytes/word). The following two words (words 1 and 2) specify the load address for this record. (The load address is the beginning location for storing this record.) The most significant load address word (bits 31 thru 16) will be used only if required. If not used, bits 31 thru 16 will contain 0000H.

The remaining words in the record (3 thru n) contain the data bytes. If the last byte in the last word of a record is not used for data, it must contain an ASCII blank (code 20H).

The format relationship between the user buffer when reading from or writing into a 64000 absolute file is also shown in Figure 8-38.

**Relocatable File (File Type 3)**

The relocatable file is produced by the assembler or compiler. It contains information required by the linker to construct an absolute file. This file consists of the following six types of records (see Figure 8-39):

- o Program Description Record (one per file)
- o Global Symbols Record
- o Data Record
- o External Symbols Record
- o Local Symbols Record (optional)
- o End Record (one per file)

Each type of record is defined in the following paragraphs.

### **Program Description Record**

(See Figure 8-40) - The program description record is the first record in the Relocatable File and only one is allowed per file. This record identifies the source program, number of externals, microprocessor, comments, and absolute code definitions.

This is a variable length record (up to 128 words) and is configured as follows:

- o One Record Identification (ID) Word
- o 14 words allocated to:
  - Source Program Name (9 characters, maximum)
  - Source Program ID (6 characters, maximum)
  - PROG Area Length (2 words, maximum)
  - DATA Area Length (2 words, maximum)
  - COMN Area Length (2 words, maximum)
- o One word allocated to definition of the number of external variables and procedures defined in the module.
- o Eight words allocated to:
  - Microprocessor Name (9 characters, maximum)
  - Microprocessor ID (6 characters, maximum)
- o Two words allocated to:
  - Date (one word, maximum)
  - Time (one word, maximum)
- o 11 words allocated to comments
- o Up to 88 words allocated to absolute code segment description.
- o One checksum word

Each of these items is described as follows:

Record identification (ID) word - The record ID word is always the first word in the record. In this case, it is also the first word in the Relocatable File and contains the number "1". This number identifies the record as the source program description record. (The "1" is used internally and should not be confused with the file type number which is "3".)

Source program name and user ID word block - An eight word block (words 1 thru 8) is allocated to contain the source program name and user ID words. This is the same ID entered into the 64000 in response to the user ID prompt. This block is always eight words long even if all words are not required to define the source program name and user ID. These eight words are constructed as follows:

- a. Word 1 - This is the first word and user ID word. The least significant eight bits (7-0) in this word contain the first ASCII character of the source program name. The most significant eight bits (15-8) always contain the following information:
  - o Source Program Name Length (PNL) - Bits 15, 14, and 13 specify the number of 16-bit words -1 used for the name. The minimum number of characters that may be used in the name is one, which requires one word. Thus, the minimum value for PNL is zero. The maximum number of characters that may be used in the name is nine, which requires five words. Thus, the maximum value for PNL is four. (See "Words 2 thru 8", below.)
  - o User ID Length (IDL) - Bits 12 and 11 specify the actual number of 16-bit words required for the user ID. (Note that IDL differs from PNL in that IDL specifies the actual number of words and PNL specifies the number of words -1.) The maximum number of characters that may be used in the user ID is six, which requires three words. Thus, the maximum value for IDL is 3.
  - o Bits 10-8 contain the number of the disc which holds the record.
- b. Words 2 thru 8 - These words are used for the remaining name and user ID characters. The name characters are specified first, followed by the user ID characters. However, name and ID characters can not be mixed within the same word. An unused least significant byte in either a name or ID word must contain an ASCII blank (Code 20H). The name and ID words must be packed. That is - the ID words, must follow the name words with no intervening unused words. Unused words must be at the end of the block.

Length word block - A six word block (words 9 thru 14) is allocated to contain the word lengths of code produced by the assembler or compiler in each of the three relocatable sections; PROG, DATA, and COMN.

Number of externals word - One word (word 15) is allocated to contain the number of external variables and procedures defined in the module. This number can be from 0 to 511.

Microprocessor name and user ID word block - This word block is the same as described for the Linker Symbols File under the "Microprocessor Configuration Record, Name and User ID Word Block".

Date and time word block - Two words (words 24 and 25) are allocated to contain the date and time that the program was assembled or compiled.

Comments word block - A block of eleven words (words 26 thru 36) is allocated for comments. The block contains up to 22 ASCII characters defined by the NAME pseudo in the assembler or compiler. All unused characters must contain ASCII blanks (code 20H).

Absolute code segment word block - A variable length block which contains from 0 to 22 entries of four 16-bit words is allocated for absolute code segments. Each four-word entry defines an absolute code segment declared in the assembler or compiler.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum word will be the 128th word. (Word #127.)

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

### **Global Symbols Records**

(See Figures 8-31 and 8-32.) - The global symbols record formatting and packing for the Relocatable File is the same as described for the Linker Symbols File under the "Global Symbols Records".

### **Data Records**

(See Figure 8-41.) - The data records contains the relocation area and address of the program as assigned by the linker. It also defines how the absolute codes are produced.

Record identification (ID) word - The ID word is always the first word in each record and contains the number "3". (The "3" is used internally and is not to be confused with the file type number, which is also "3".

Relocation address words - These words contain the relocation address assigned by the linker to this program. The most-significant word is used only when the ID offset equals 3.

Relocation word - The relocation word identifies the relocation destination code as follows: 00=ABS, 01=PROG, 10=DATA, and 11=COMN.

Event selection word - This word contains codes 00, 01, 10, and 11 in bit locations T1 thru T8. Any one of the codes may be contained in any of the locations. As T1 thru T8 are read, the event selected by the specific code will be executed. Codes are defined as follows:

- Tn=00 Produce one byte of absolute code, which is found in the low order byte of the corresponding word.
- Tn=01 Produce two bytes of absolute code, which is found in the corresponding word.
- Tn=10 Relocate the address to be found in the second (and optionally, the third) word based on the relocation code in the first word. Then produce an absolute code based on the processor dependent format number in the first word and skeleton, if used.
- Tn=11 Look up the external symbol whose number is in the first word (which has been previously defined in a type 4 record). Add the displacement and then produce an absolute code based on the format number and skeleton, if used.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

### **External Symbols Records**

(See Figure 8-42.) - The Relocatable File may contain multiple External Symbols Records.

An External Symbols Record contains the external symbols and the external ID number assigned by the assembler or compiler. Each record may consist of up to 128, sixteen-bit words (words 0-127) structured as follows:

- o One Record Identification (ID) Word
- o Multiple External Symbol Definition Blocks
- o One Checksum Word

Each of these items is described as follows:

Record identification (ID) word - The ID word is always the first word in each record and contains the number "4". (The "4" is used internally and is not to be confused with the file number, which is "3".)

External symbol definition blocks - An external symbol definition block consists of the symbol word(s) and the external ID number. (See Figure 8-42.)

Symbol words - The ASCII character, or characters, are contained in this word, or words. From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated to 15 characters.)

First symbol word - The first word in every symbol definition block is structured the same. The least significant 8 bits (7-0) contain the first ASCII character in the symbol. The most significant eight bits (15-8) always contain the following information:

- o External Symbol Length (ESL) - Bits 15, 14, and 13 specify the number of symbol words -1 in this block. For example, if the external symbol consists of two ASCII characters, which requires two symbol words, then ESL is equal to 1. (The second byte in the second word will contain an ASCII blank - i.e. code 20H)
- o Reserved Bits - Bits 12, 11, 10, 9, and 8 always contain 00100.

Additional symbol words - The second thru the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (code 20H). That is, the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words are required to define a symbol, then only five words are to be used.

External ID number word - The external ID number is assigned by the assembler or compiler. The number can be from 0 to 511.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

### **Local Symbols Records**

(See Figures 8-31 and 8-32.) - The local symbols records formatting and packing for the Relocatable File is the same as described for the Linker Symbols File under the "Global Symbols Records", except the ID word contains the number "6".

**End Record**

(See Figure 8-43.) - The end record is the last record in the Relocatable File and only one is allowed per file. The end record contains the relocation code and transfer address. Each record consists of five, 16-bit words structured as follows:

- o One Record Identification (ID) Word
- o One Relocation Word
- o Two Transfer Address Words
- o One Checksum Word

Each of these items are described as follows:

Record identification (ID) word - The ID word is always the first word in each record and contains the number "5". (The "5" is used internally and is not to be confused with the file number, which is "3".)

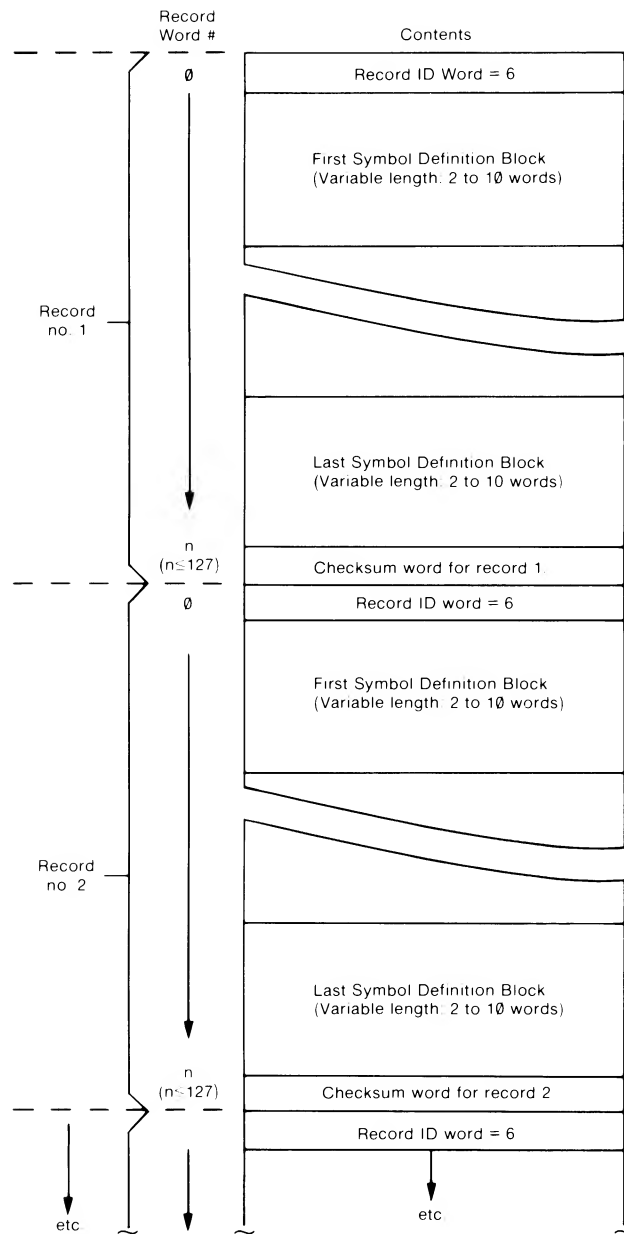
Relocation word - The relocation word identifies the relocation destination code, as follows: 00=ABS, 01=PROG, 10=DATA, and 11=COMN.

Transfer address words - The transfer address words contain the address where control will be transferred to when the program is run.

Checksum word - The checksum word must be the last word in each record. The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

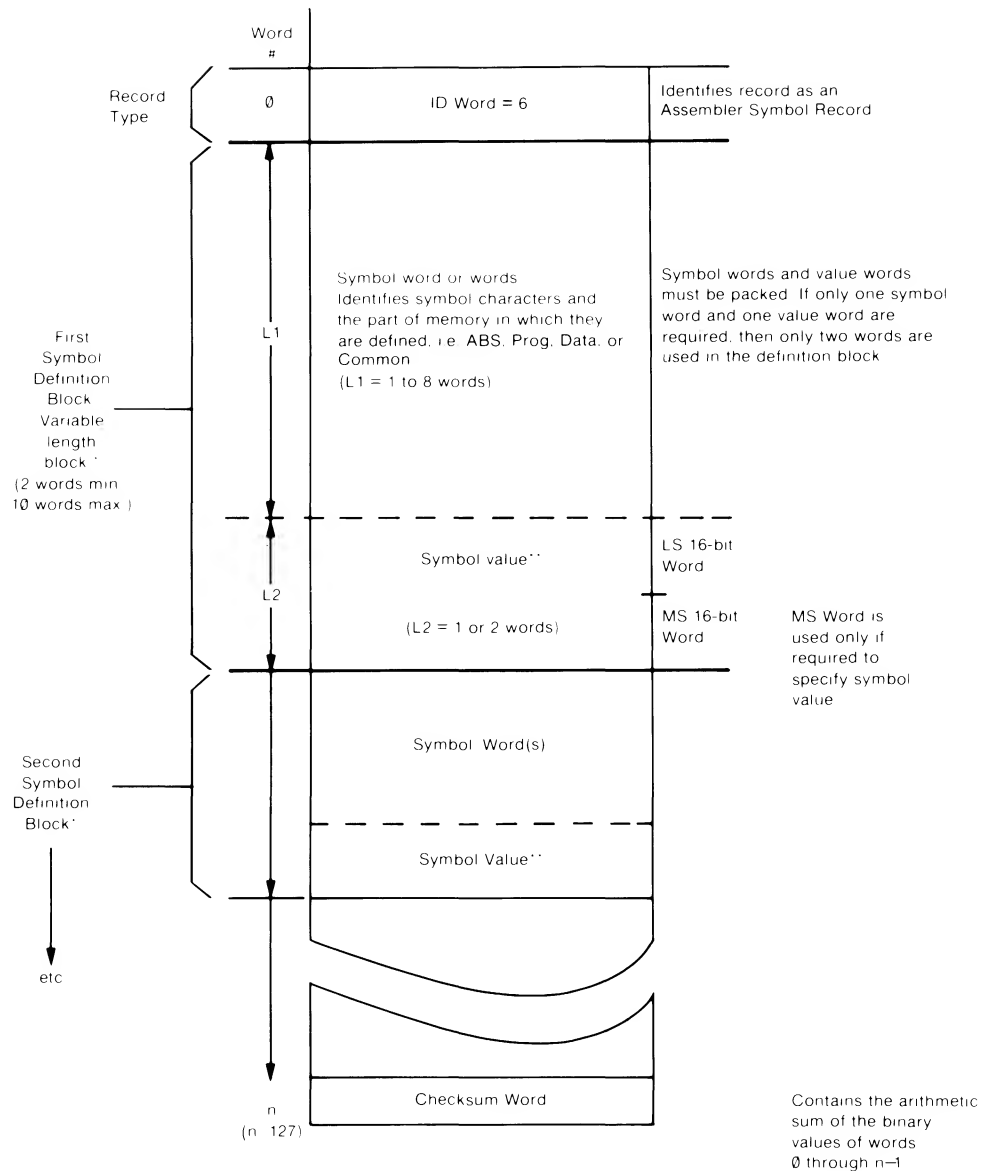
**User Buffer/Relocatable File Packing Formats**

The format relationship between the user buffer when reading from, or writing into, a 64000 Relocatable File is the same as shown for the Assembler Symbols File in Figure 8-27.



**Figure 8-25. Assembler Symbol File Overall Structure**

## ASSEMBLER SYMBOL RECORD STRUCTURE

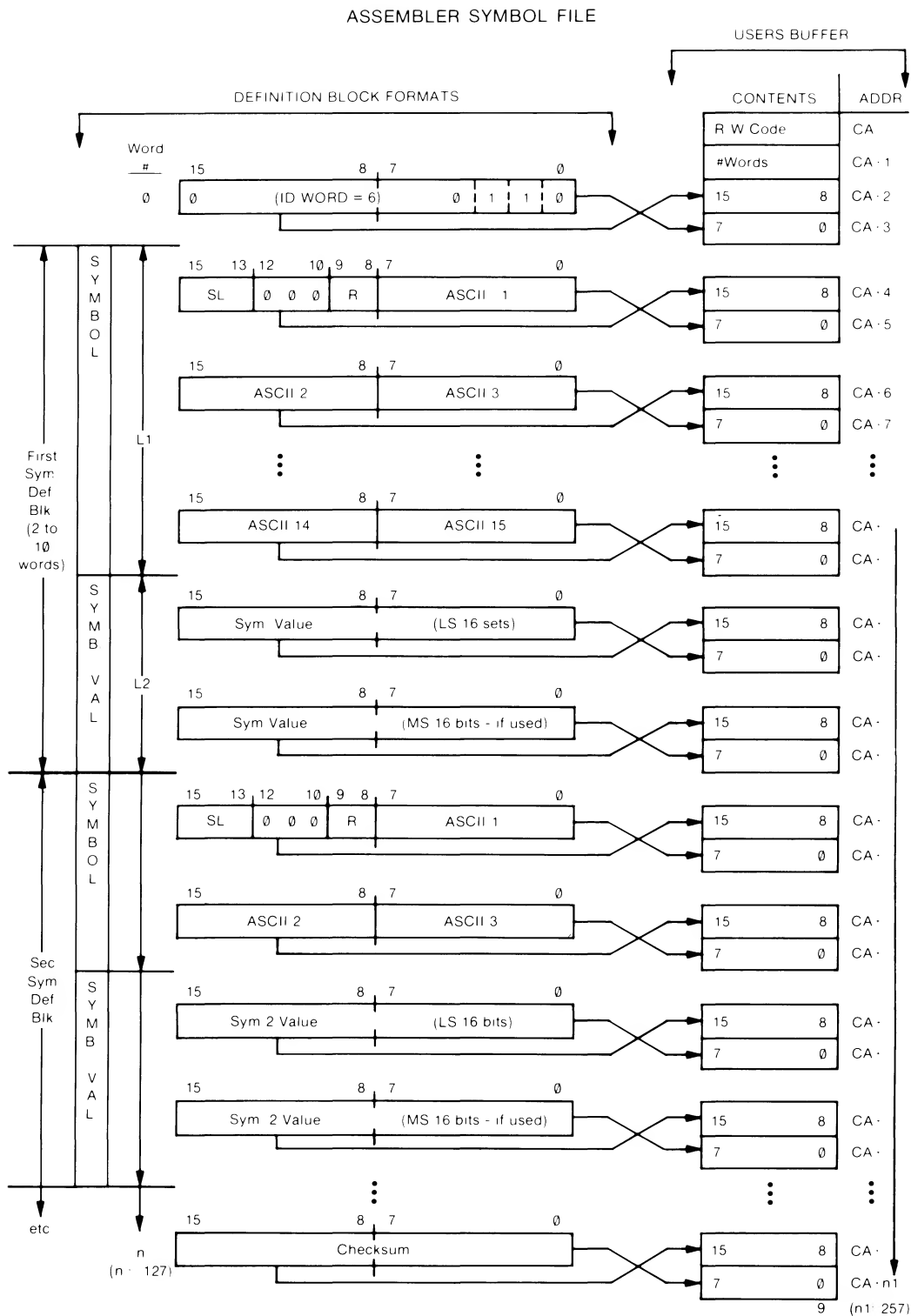


### Notes

\*For block structure details, see "Assembler-Symbol Record/User Buffer Format Details".

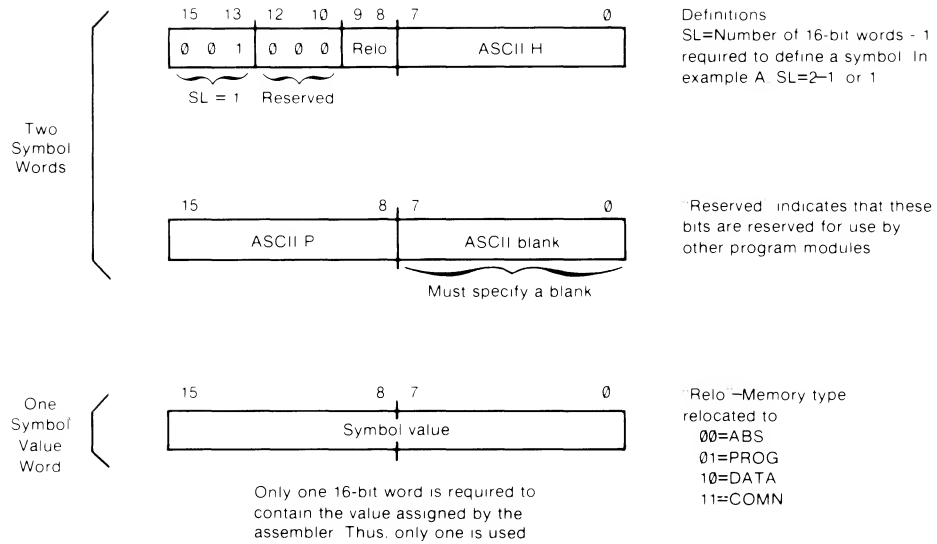
\*\*Symbol value as assigned by assembler. If a relocatable value it will be relocated by the linker.

**Figure 8-26. Assembler Symbol Record Structure**

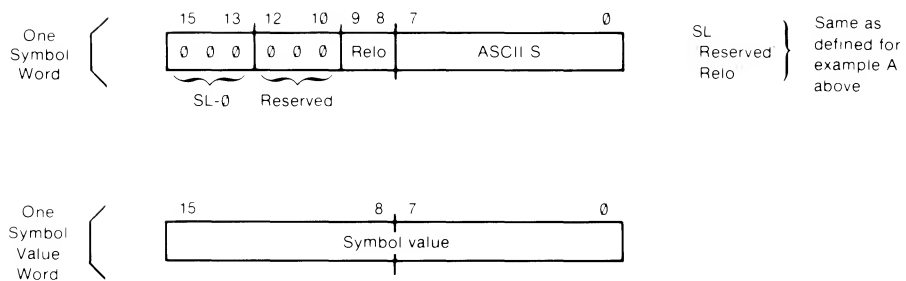


**Figure 8-27. Assembler Symbol Record/User Buffer Format Details**

EXAMPLE A. SYMBOL = HP

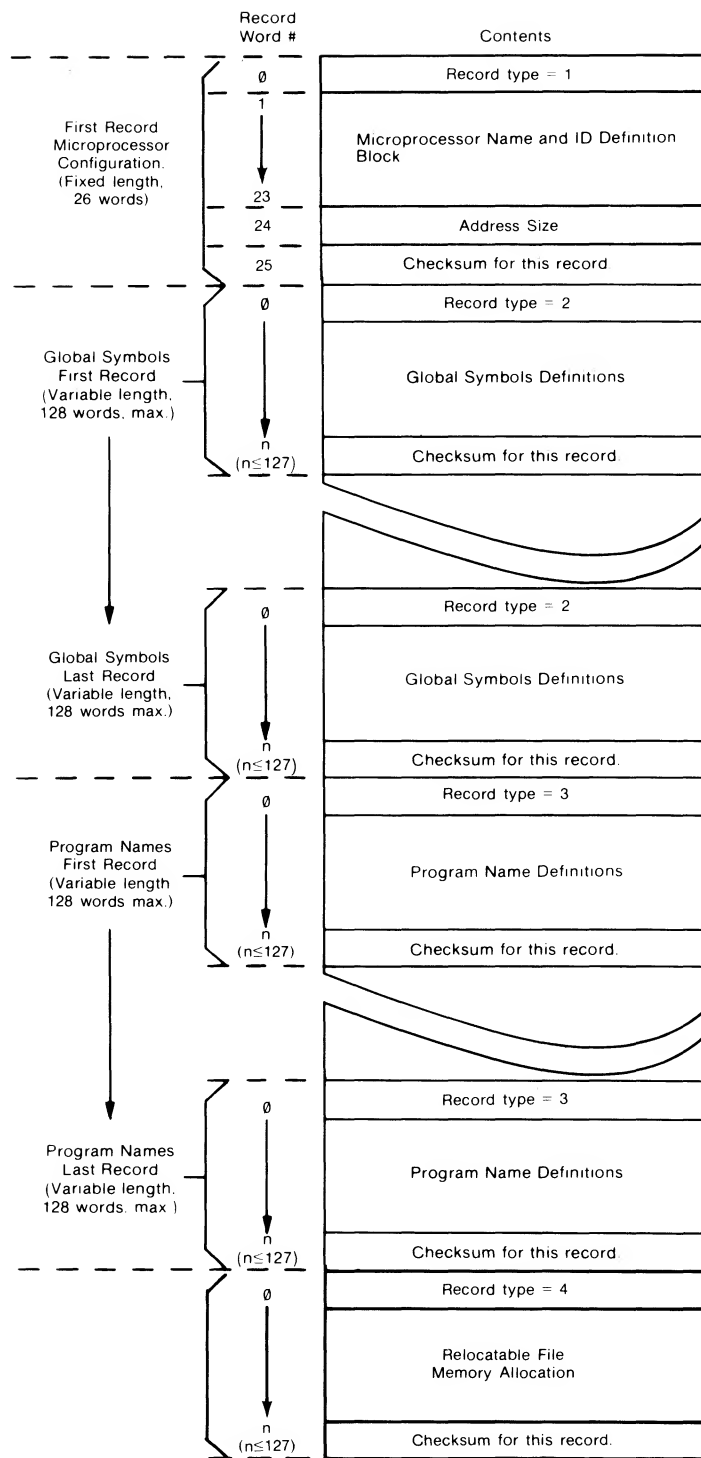


EXAMPLE B. SYMBOL = S

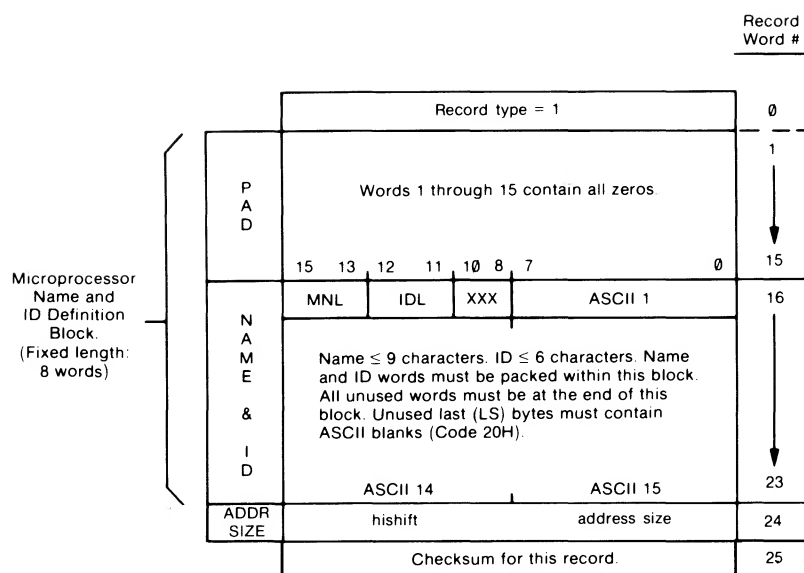


Again, only one 16-bit word is required to contain the symbol value. Thus, only one is used.

**Figure 8-28. Assembler Symbol Record/Symbol Definition Block Examples**



**Figure 8-29. Linker Symbol File Overall Structure**

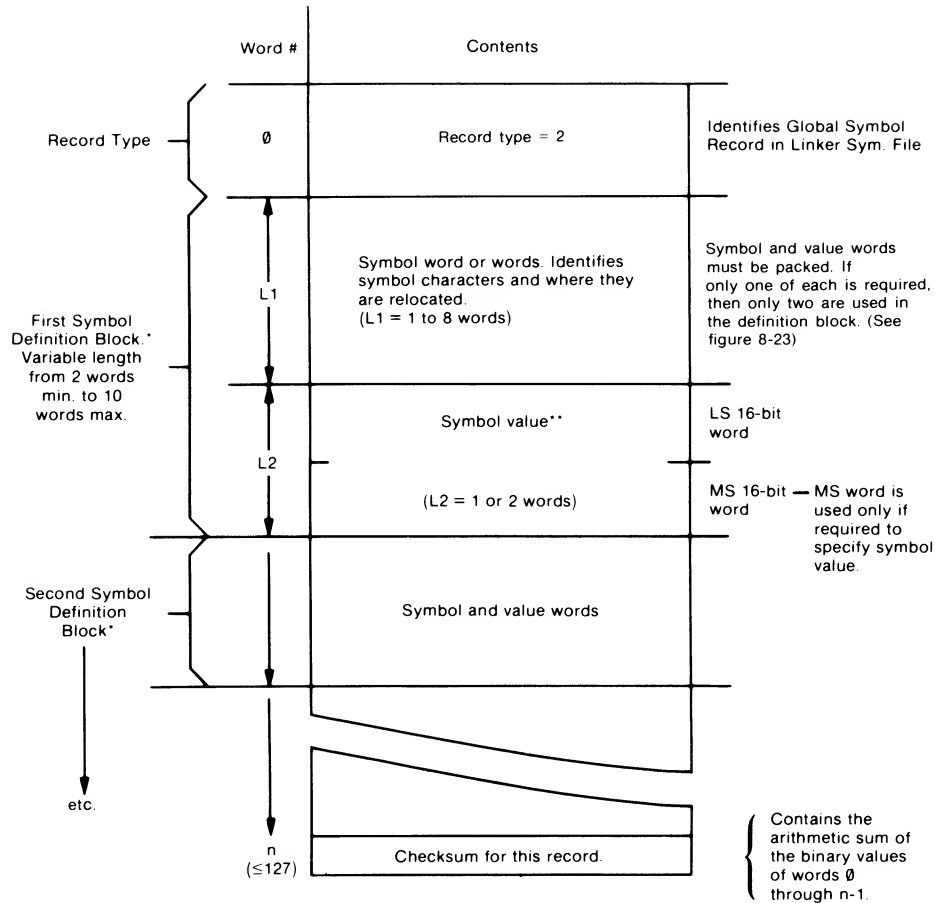


#### Notes

1. Words 1 through 15 are added so that word positions 16-23 in this name record contain the same data as do the corresponding word positions in the name records of the relocatable files.
2. MNL = Number of 16-bit words - 1 required to define the microprocessor name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, MNL = 0. If all nine characters are used (5 words), MNL = 4.
3. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used, IDL = 3.
4. Bits 10, 9, and 8 - "don't care" condition.
5. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain an ASCII blank (Code 20H).
6. Word 24 indicates address size where "address size" = 1 indicates one-word addresses (16 bits) and "address size" = 2 indicates two-word addresses (32 bits). "hishift" used when converting logical addresses (segment, offset) to physical addresses.
7. The checksum contains the arithmetic sum of the binary values of words 0 through 24.

**Figure 8-30. Microprocessor Configuration Record Structure**

## GLOBAL SYMBOLS RECORD

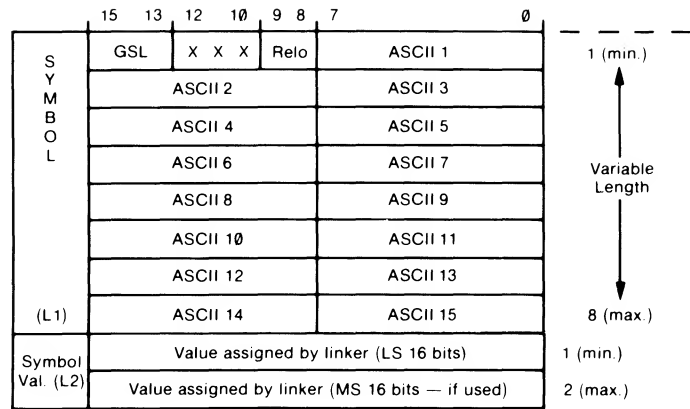


### Notes

\*For block structure details see "Global Symbols Definition Block Diagram."

\*\*Symbol value assigned by assembler. If relocatable value (not ABS), it will be relocated by the linker.

**Figure 8-31. Global Symbol Record Structure**

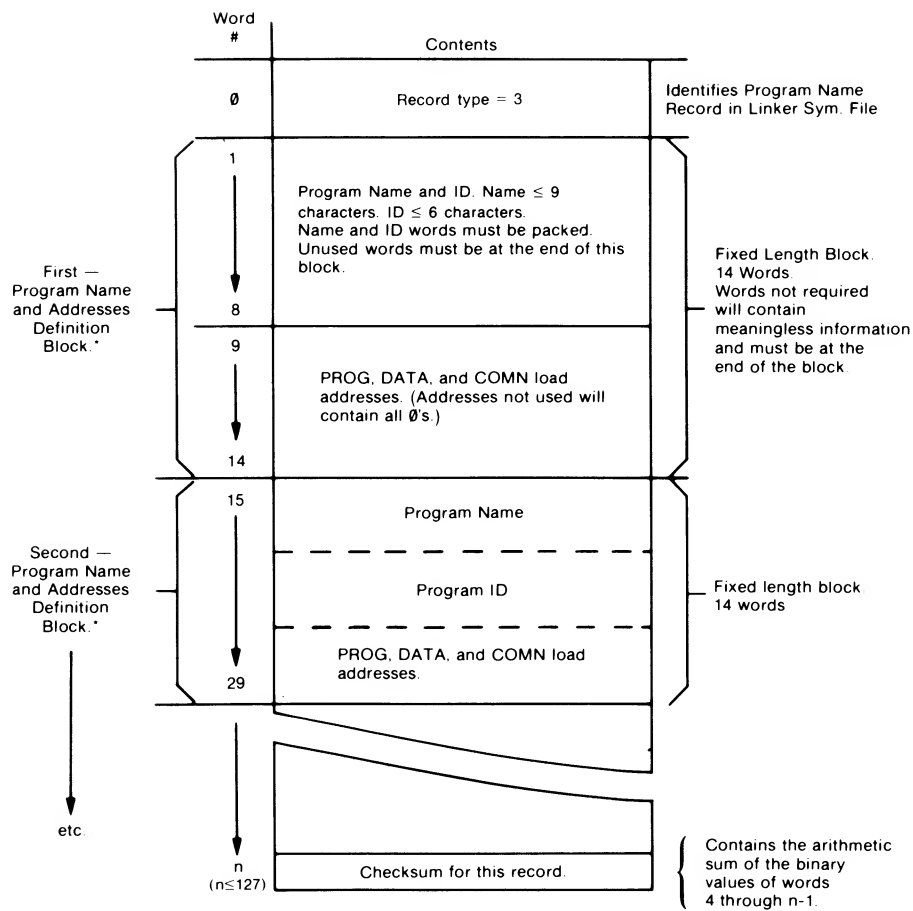


#### Notes

1. GSL = Number of 16-bit words - 1 required to define a global symbol. At least one character is required in the "ASCII 1" byte. Thus, with a one character name, name length = 0. If all 15 characters are used (8 words), name length = 7.
2. Bits 12, 11, 10 - "don't care" conditions.
3. "Relo" contains the binary code for area relocated to as follows: 00 = ABS, 01 = PROG, 10 = DATA, and 11 = COMN.
4. The bytes labeled ASCII 1-15 are the maximum number of bytes available to define the symbol. Only the actual number of 16-bit words required to define the symbol will exist. However, if the first byte (MSB) is used, then the second byte (LSB) must contain an ASCII blank (Code 20H).
5. The symbol value is assigned by the assembler. If a relocatable value it will be relocated by the linker. The 8086 microprocessor symbol values are in segment, offset form where LS = offset and MS = segment.

**Figure 8-32. Global Symbol Definition Block**

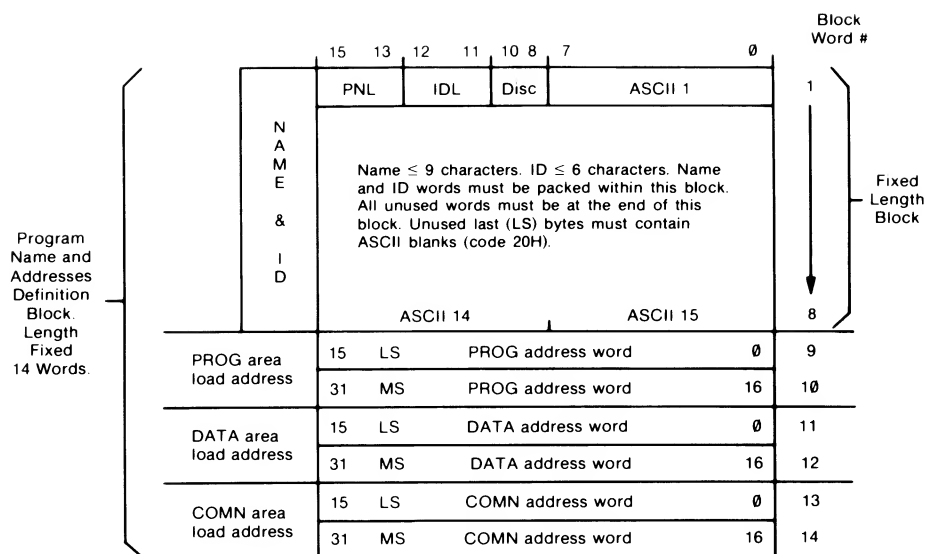
# PROGRAM NAME RECORD



## Notes

\*For block structure details, see figure 8-34.

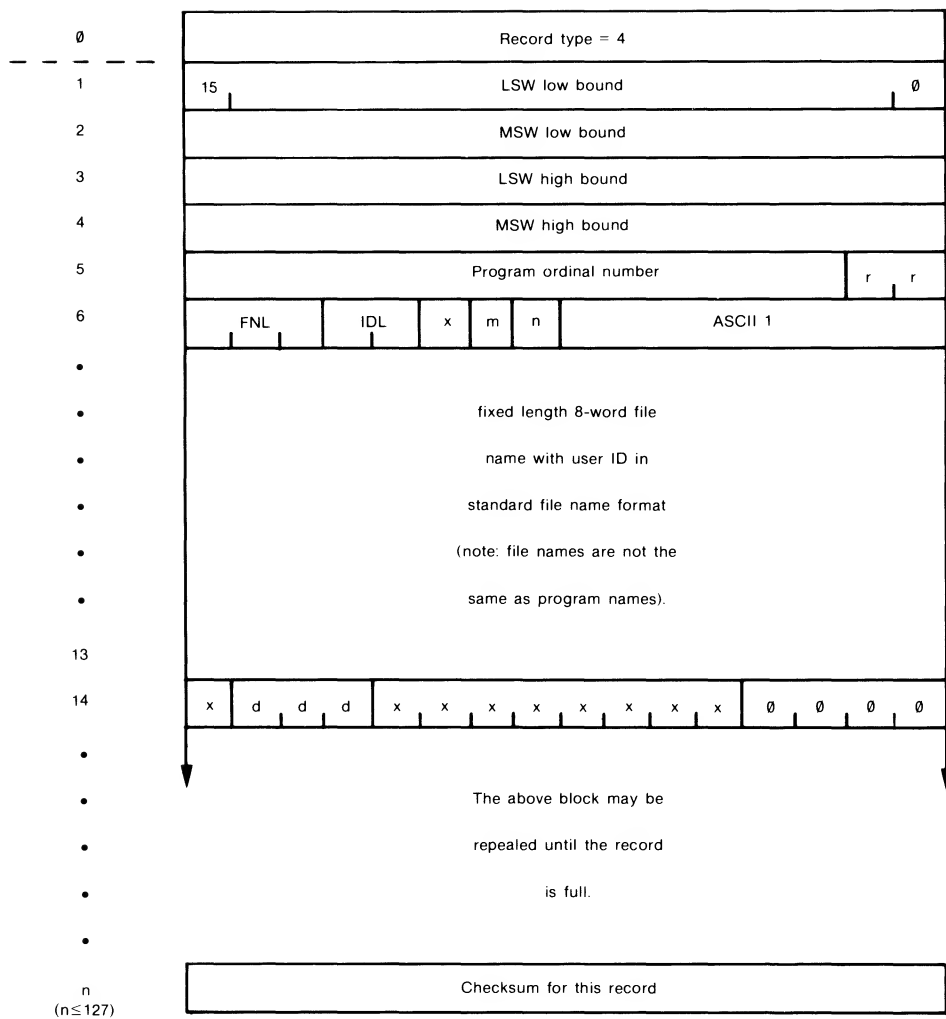
**Figure 8-33. Program Name Record Structure**



#### Notes

1. PNL = Number of 16-bit words - 1 required to define the program name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, PNL = 0. If all nine characters are used (5 words), PNL = 4.
2. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used, IDL = 3.
3. DISC = The indentifying number of the disc upon which the program resides.
4. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is - the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain and ASCII blank (Code 20H).
5. Load Address Words - The load address words contain the load address assigned by the linker to this program. Unused address words contain all zeros. Load addresses for the 8086 microprocessor are in segment, offset form where LS = offset and MS = segment.

**Figure 8-34. Program Name and Address Definition Block Format**



**Figure 8-34A. RANGE Definition Block Format**

**Notes (for Figure 8-34A)**

1. Words 1 through 4 list the memory space used by the relocatable files. Blocks are sorted on lower bound from smallest to largest. For the 8086 microprocessor, bounds are in offset, segment form where LSW = offset and MSW = segment.
2. Word 5 - a Program name for TYPE 3 records has an implicit ordinal number value which is indicated in word 5. The two bits, rr, indicate program type as follows:

rr = 00 → absolute  
rr = 01 → PROG relocatable  
rr = 10 → DATA relocatable  
rr = 11 → COMN relocatable

3. Word 6 - FNL = number of 16-bit words-1 required to define the file name.

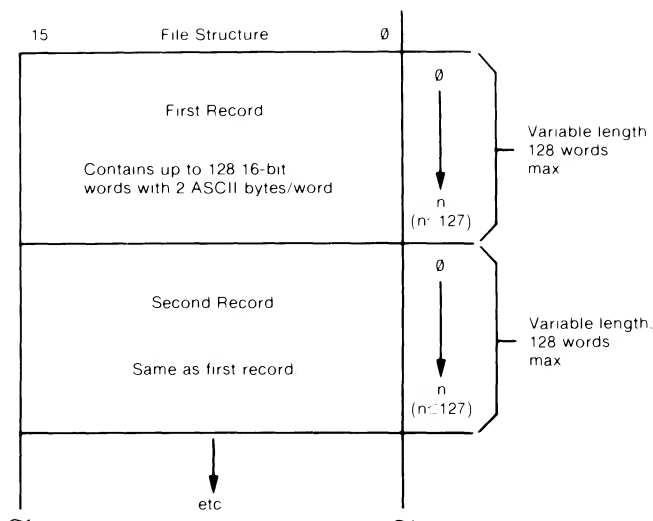
IDL - actual number of 16-bit words required to define user ID. If one word is used, IDL = 1.

x - indicates "don't care" condition.

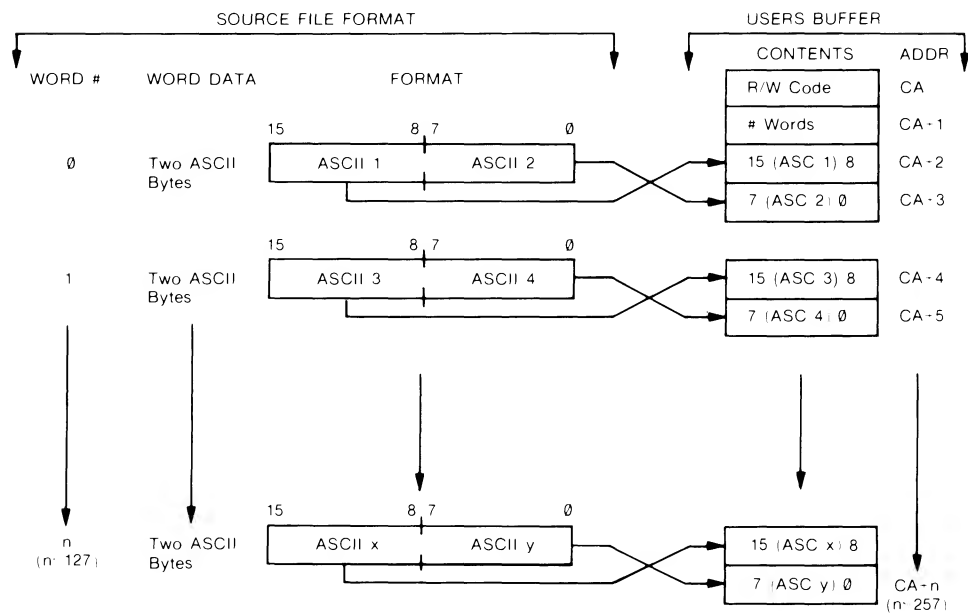
m - as described under TYPE 3 RECORDS, a file may contain multiple relocatables or a program name different from the file name. If this occurs, m = 1. If m = 0, the file name and program name are the same.

n - If n = 1, the indicated file was a "no-load" file, i.e., the file was linked and relocated but no code was generated.

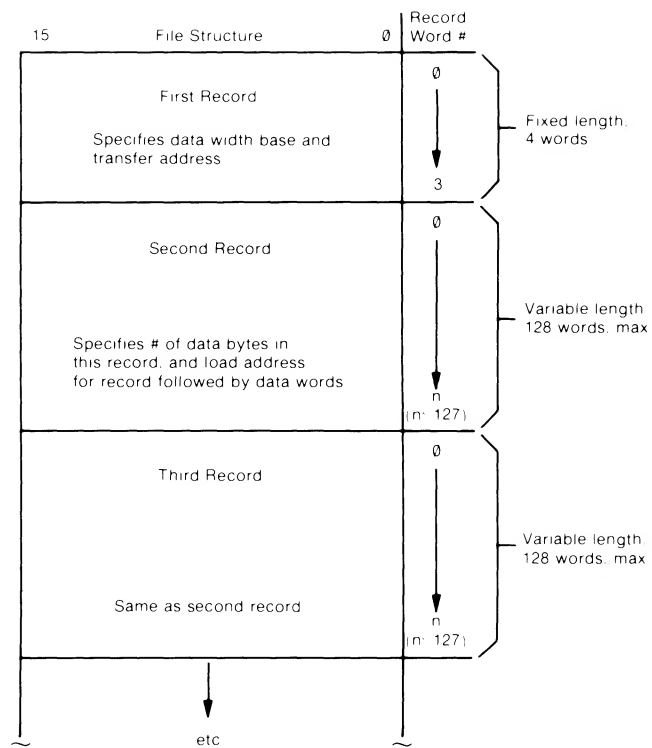
4. Word 14 - ddd indicates the disc where the file resides. x = "don't care" conditions.



**Figure 8-35. Source and Listing Files - Overall Structure**

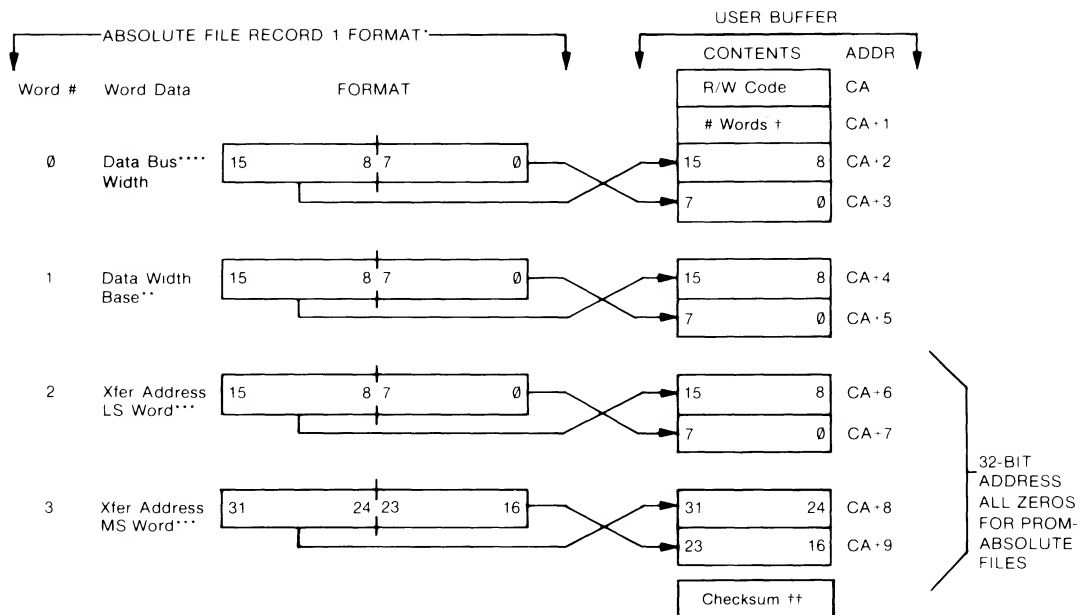


**Figure 8-36. Source and Listing File Format**



**Figure 8-37. Absolute File - Overall Structure**

ILLUSTRATION A.  
RECORD 1 FORMAT ONLY.  
(Format for all Other Records Shown on Illustration B)



**Notes**

\*Record 1 must precede all other records in an absolute file and it must always be formatted as shown. (Always four words.)

\*\*The Data Width Base is the minimum addressable entity (i.e., group of bits) used by the microprocessor. Normally this will be 8 bits but not always.

\*\*\*The transfer address is the value loaded into the microprocessor program counter. This value is all zeros for PROM Absolute files.

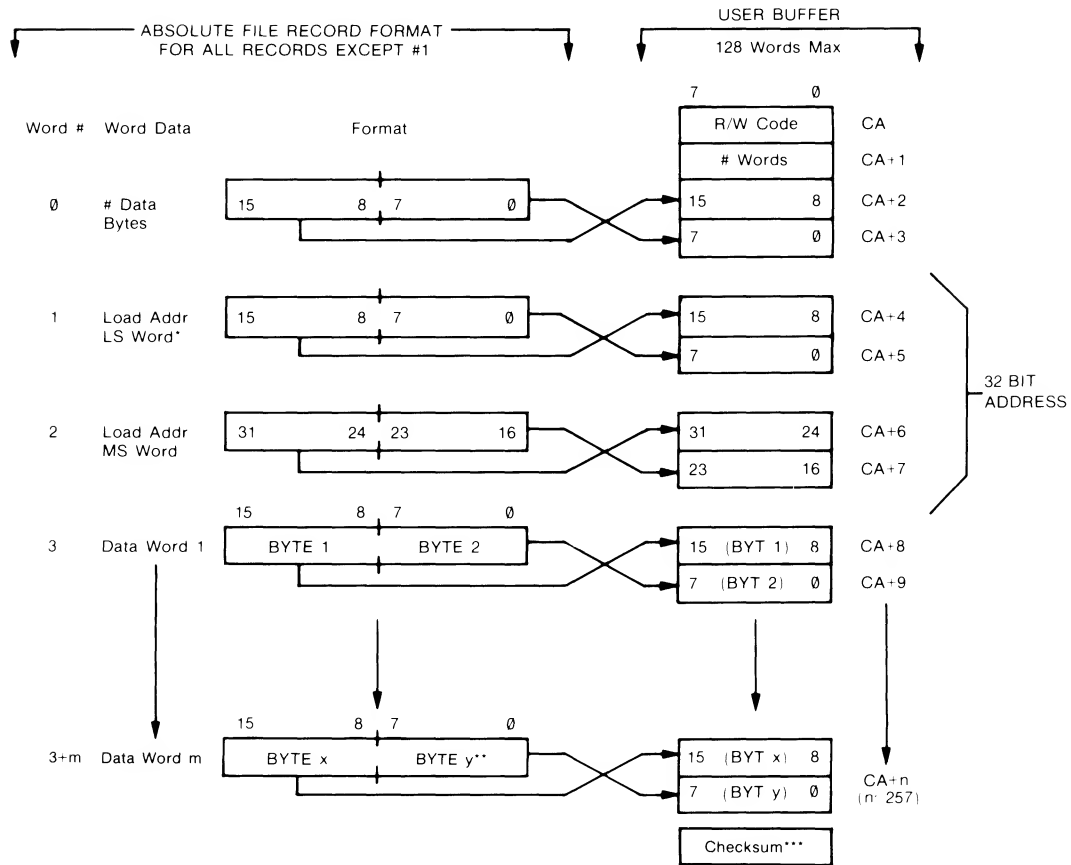
\*\*\*\*Width of processor data bus (i.e., 8, 16 etc.)

† Total number of words in record excluding checksum and number of words, (i.e. n-2), always equal to 4 for record 1.

†† The checksum is the module 256 sum of bytes CA+2 through CA+9.

**Figure 8-38. Absolute File Formats**

ILLUSTRATION B.  
FORMAT FOR ALL RECORDS EXCEPT RECORD 1  
(See Illustration A for Record 1 Format)



**Note**

\*The load address is the address of the first location into which this record is stored.

\*\*This last byte will be a pad byte if the record contains an odd number of bytes. This is required to fill up the word boundary.

\*\*\*The checksum is the module 256 sum of bytes CA+2 through N-1.

**Figure 8-38. Absolute File Formats (Cont'd)**

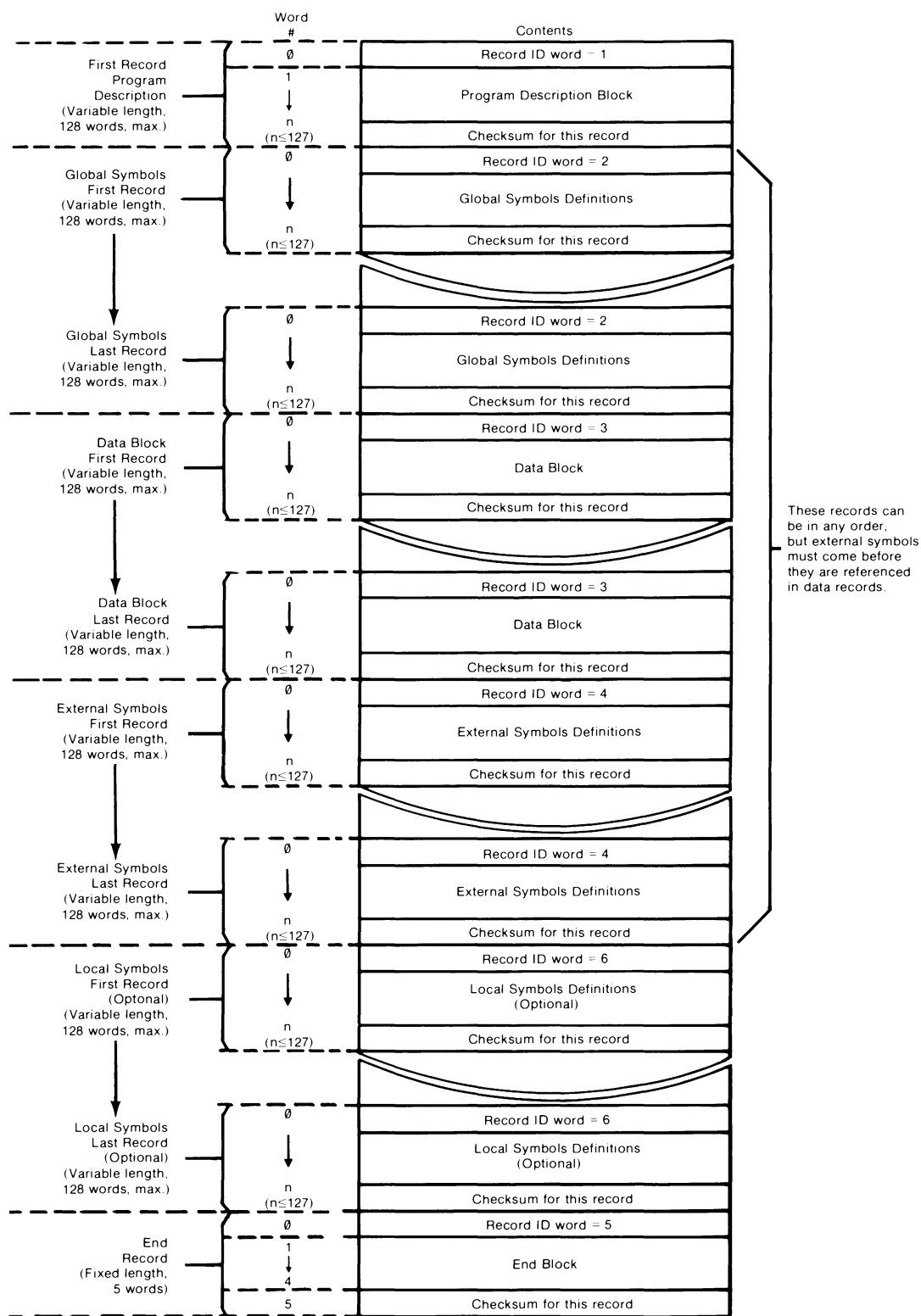
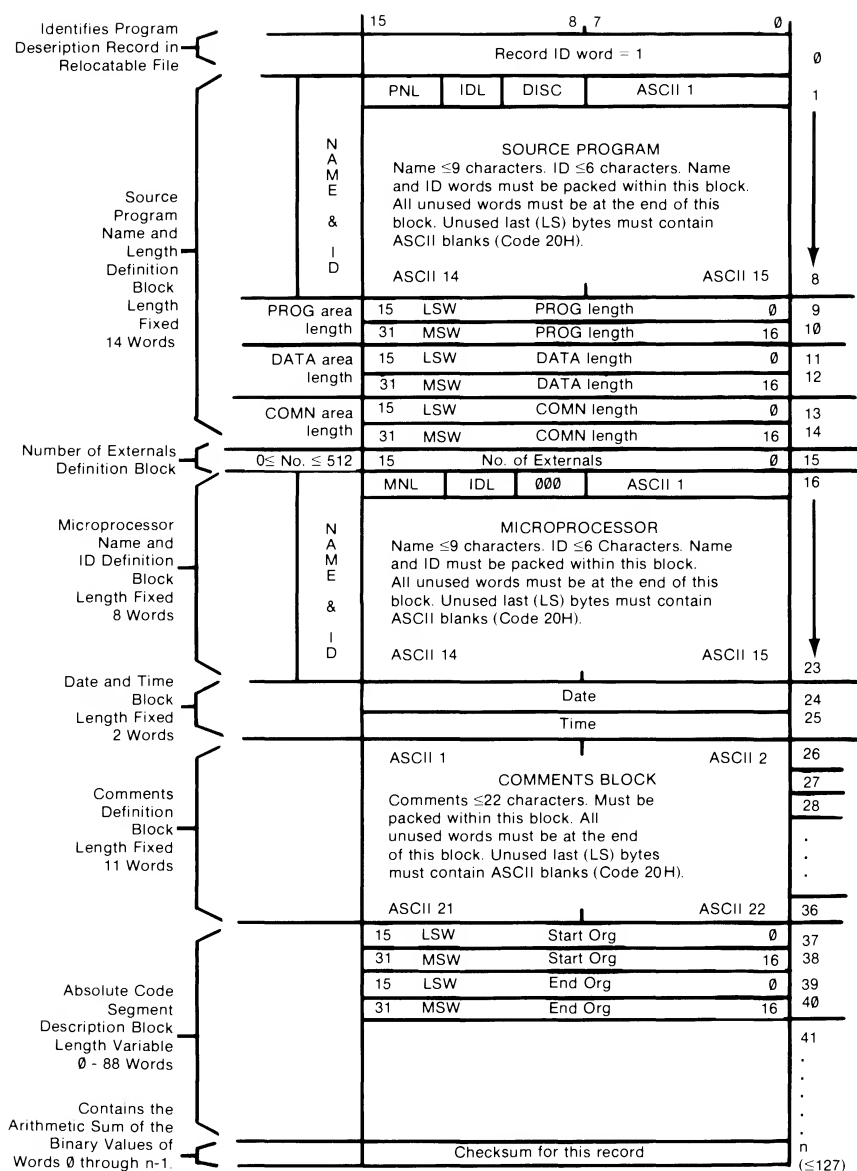


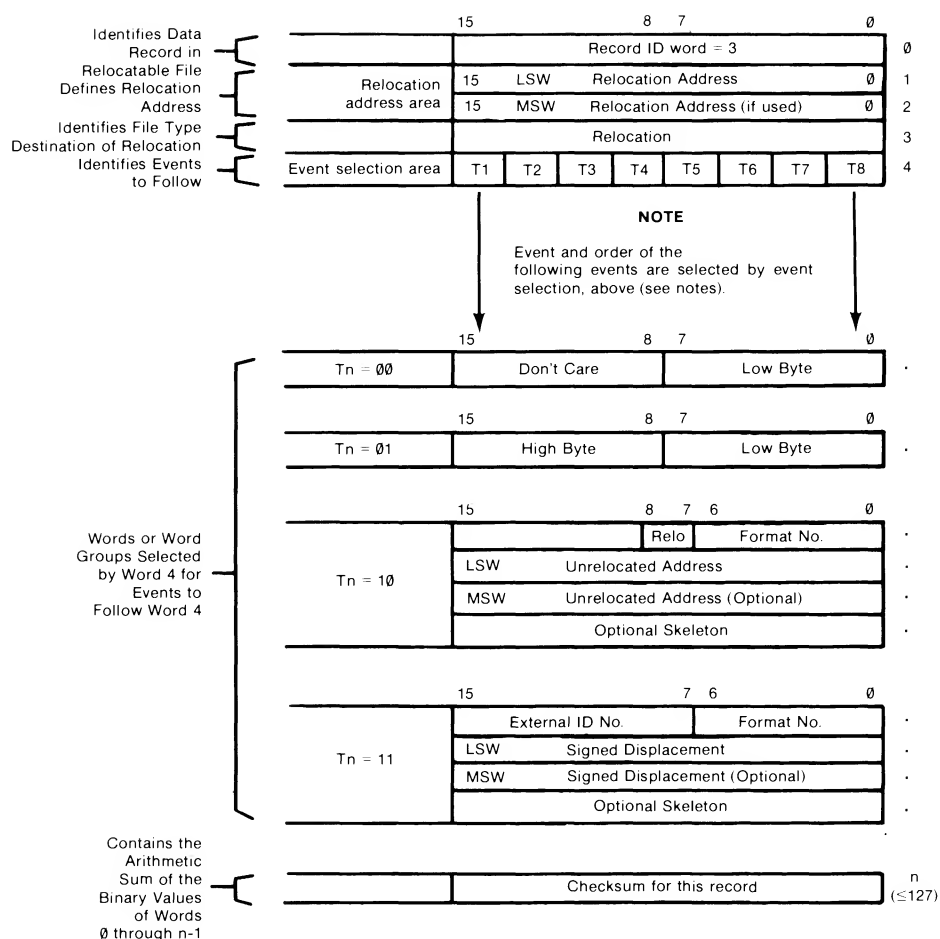
Figure 8-39. Relocatable File Overall Format



#### NOTES:

1. PNL and MNL = Number of 16-bit words-1 required to define program or microprocessor name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, PNL or MNL = 0. If all nine characters are used (5 words) PNL or MNL = 4.
2. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used IDL = 3.
3. Disc (in program name segment) - The identifying number of the disc upon which the program resides.
4. Bits 10, 9, and 8 in microprocessor name segment always contain 000.
5. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is; the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain an ASCII blank (Code 20H).
6. Length bytes or words - Contains the number of bytes or words (processor dependent) of code produced by the assembler or compiler in each of the three relocatable sections; PROG, DATA, COMN.
7. Number of externals - Contains the number of external variables and procedures defined in the module.
8. Comments - Contains up to 22 ASCII characters defined by the NAME pseudo in the assembler or compiler. All unused characters must contain ASCII blanks (Code 20H).
9. Absolute code segment description - Contains 0 to 22 entries of four 16-bit words. Each four word entry defines an absolute code segment declared in the assembler or compiler.

**Figure 8-40. Relocatable File Program Description Definition Block**

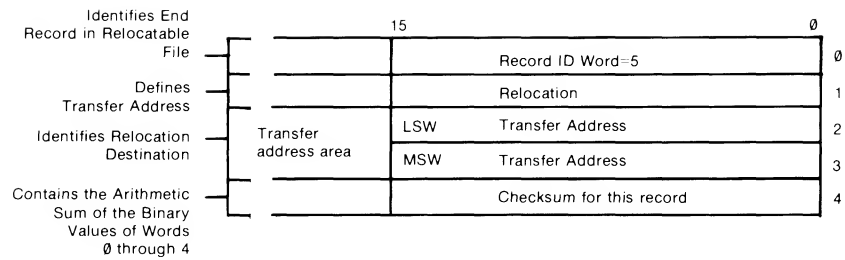


#### NOTES:

1. Relocation Address Words - The relocation address words contain the relocation address assigned by the linker to this program. The MSW is used only when the ID offset = 3.
2. Relocation contains the binary code for area relocated to as follows: 00 = ABS, 01 = PROG, 10 = DATA, and 11 = COMN.
3. Event Selection Area - Selects events to follow. T1 through T8 may contain any one of codes 00, 01, 10, or 11. Codes are defined as follows: 00 = one byte absolute with no modifications, 01 = two bytes absolute with no modifications, 10 = relocatable reference, and 11 = external reference. As T1 through T8 are read, the event selected by the specific code will be executed.
4. Tn = 00 - Produce one byte of absolute code, which is found in the low order byte of the corresponding word.
5. Tn = 01 - Produce two bytes of absolute code, which is found in the corresponding word.
6. Tn = 10 - relocate the address to be found in the second word (and optionally, the third word) based on the relocation code in the first word. Then produce an absolute code based on the processor dependent format number in the first word and skeleton, if present.
7. Tn = 11 - look up the external symbol whose number is in the first word (which has been previously defined in a type 4 record). Add the displacement and then produce an absolute code based on format number and optional skeleton.

**Figure 8-41. Relocatable File Data Definition Block**





**Notes:**

1. Relocation-contains the binary code for area relocated to as follows: 00=ABS, 01=PROG, 10=DATA, 11=COMN, 100=No transfer address.
2. Transfer Address Words-Contains the address where control will be transferred to when the program is run. Only one module in a program may have a transfer address, and it is defined in the END label psuedo in the assembler or the presence of the main program block in a PASCAL module.

**Figure 8-43. Relocatable File End Definition Block**



## Syntactical Variable Definitions

The syntactical variables used throughout this manual are described in this appendix.

### **<ABSFILE>**

The <ABSFILE> is the file identifier of an absolute file that contains the emulation program. The emulation program is placed into the file by assembling and linking to the file before application to the target microprocessor. <ABSFILE> has the same format requirements as the <FILE> variable which is described later in this appendix.

### **<ADDRESS>**

The <ADDRESS> variable defines a bit pattern of up to 16 bits which specifies a particular location in mapped memory. That bit pattern can be represented by a binary, octal, hexadecimal, or decimal number; a local or global symbol; or a mathematical combination of numbers or symbols. <ADDRESS> has the same format requirements as the <VALUE> variable which is described later in this appendix.

### **<ADR\_LST>**

The variable <ADR\_LST> contains a list of addresses, separated by commas, where the addresses are within the address space defined by the processor.

### **<CMDFILE>**

The <CMDFILE> variable is the file identifier for an existing emulation configuration file. This command file contains the organizational commands for the processor to be emulated. The command file can be retained or modified for further use. <CMDFILE> has the same requirements as the <FILE> variable which is described later in this appendix.

### **<FILE>**

The <FILE> variable is used to identify files generated or accessed by the development system commands. <FILE> consists of the following parameters:

**<FILE NAME>[:<USERID>][:<DISC#>]**

where:

<FILE NAME>	is the identifier given to a particular file. <FILE NAME> must begin with an upper case alphabetic character and can have a total length of nine characters. After the first character, any upper or lower case alphanumeric character or an underscore can be used. If more than nine characters are specified, the name is truncated to the first nine characters.
<USERID>	is the identifier assumed by a particular system user. <USERID> must begin with an upper case alphabetic character and can have a total length of six characters. The characters following the first character can be any upper or lower case alphanumeric characters, including the underscore. If more than six characters are specified, the userid is truncated to the first six characters. If a userid is not entered, the current userid is used as the default.
<DISC#>	specifies the disc on which the file is stored. <DISC#> can be any digit from 0 thru 7, but it must correspond to the Logic Unit number assigned to one of the discs at system power up. The default is to search the discs for the file specified, or to create the file on disc zero.

### <REAL\_VAL>

The <REAL\_VAL> variable is an alphanumeric representation of a real number value. The syntax is:

$$\left\{ \begin{array}{l} \left\{ \begin{array}{c} + \\ - \end{array} \right\} <\text{integer}> \\ \left\{ \begin{array}{l} <\text{integer}> \quad \left[ E \left\{ \begin{array}{c} + \\ - \end{array} \right\} <\text{integer}> \right] \\ E \left\{ \begin{array}{c} + \\ - \end{array} \right\} <\text{integer}> \end{array} \right\}$$

Where <integer> is an unsigned decimal integer.

### <STATE>

The <STATE> variable specifies a particular state on the emulation bus. The <STATE> expression consists of an address, a data, and a status specification.

### <VALUE>

<VALUE> is a syntactical variable that allows specification of symbols (labels), numbers, parentheses, and math operators (+, -, /, (), \*) following standard algebraic rules to produce a value. Legal operands are defined in the following paragraphs.

<NUMBER> is an alphanumeric representation of a 16 bit pattern of ones, zeros, and don't cares (X's). The bit pattern can be represented in binary, octal, hexadecimal, or decimal where binary is indicated by a "B", octal by a "Q", hexadecimal by an "H", and decimal by a "D". Decimal is the default value and the use of "D" is optional.

Examples:

```
(A+B)*C
10101011XXXXXXB
145XXXQ
2563
```

The <LOCAL SYMBOL> variable represents the name of a symbol which can only be used by the program module in which it is defined. The <GLOBAL SYMBOL> variable represents the name of a symbol which can be called by program modules other than the one in which it is defined. The global symbol must be declared as such by a GLB statement in the source file.

<LOCAL SYMBOL> is specified as: SYMBOL\_NAME [:<MODULE>] or: #<LINE #> [:<MODULE>] where <MODULE> is the same as <FILE>. For PASCAL programs, lines which generate object code produce local line # symbols corresponding to the source line.

<GLOBAL SYMBOL> is specified as <SYMBOL\_NAME> or @:<MODULE> which produces the starting address of the specified <MODULE>.

<MODULE> specifies the file in which the local symbol is defined. If no <MODULE> is specified, the global symbol table associated with the absolute program file loaded by the emulator is searched for the <SYMBOL\_NAME>. If the symbol name is not found in the global symbol table, a search is made of the last referenced local symbol table. If the symbol name is not found in the local symbol table, an error message is displayed on the status line. For more information, refer to the description of <FILE> which is included in this appendix.

<STRING> is an ASCII string delimited by ', ', ^., and produces a 16 bit code.

Examples:

```
'A' 0041H
"AB" 4142H
^BC^ 4243H
```



## **6800/6802 Status and Error Messages**

### **Status Messages**

Access to guarded memory, address 0XXXXH

Guarded memory is accessed by the 64000 station through display memory or modify memory commands. 0XXXXH is the address in guarded memory.

Break in background

A break has occurred and the emulator processor is executing in the background program. See Chapter 2 for details on “break”.

Break unknown state

Control of the emulator processor is lost. A reset command should be issued to recover the processor.

halt

The halt line is pulled low by the target system (\*HALT = 0).

No memory cycles

The processor has not done a valid memory cycle during the last 500ms.

Reset in background

A reset command has been issued by the 64000. The emulator processor is in background.

Reset unknown state

Control of the emulator processor is lost. A reset command should be issued to recover the processor.

User reset

The reset line is pulled low by the target system (\*RESET = 0).

#### Running

The emulator processor is running in foreground. See Chapter 2 for details on "foreground".

#### Step complete

Single-stepping was successfully completed.

#### Step in process

The emulator is single-stepping through target program.

#### Three state control

The hardware signal TSC is active. (Applies only to the 6800 microprocessor.)

#### wait

WAI instruction was executed, the emulator's processor is waiting for an interrupt (\*BA = 1).

## Error Messages

#### Command causes break, runs restricted to real-time

If the emulator is running and 'restrict to real-time only' was specified in the configuration, commands that will cause the emulator to alternate between target program and background program are not allowed, i.e., display registers, modify memory, etc. See Chapter 4 for details on real-time restrictions.

#### Command not allowed, processor not in background

The command requires the emulator processor to be in background. An attempt has been made to break the processor, but was not successful. The emulator will recover to a "Break in background" state once the break has succeeded.

#### Illegal memory access PC=0XXXXH

An illegal memory access by the emulator processor has occurred during execution of user code (write to ROM or access to guarded memory). PC=0XXXXH is the address of the last opcode to be executed by the emulator processor before the illegal memory access. This type of error detection is possible only when a memory control board is part of the emulation subsystem.

#### Illegal opcode 0XXH at 0XXXXH

An illegal opcode was executed by the emulator processor. The opcode and the opcode address are displayed in the message.

## Radio Frequency Interference

With an emulation system installed in the Model 64000, several methods of operation (physical setup) may result in an increased emission of radio frequency noise. To reduce the r.f. noise level, any of the following techniques may be used:

- a. When the emulator is used infrequently, disconnect the emulator pod and cables from both the host system and target system.
- b. For systems that use the emulator intermittently, select "external clock" and disconnect the pod cable from the target system when not in use.
- c. Consistent with design needs, minimize the time that the emulator is used without being connected to a target system.
- d. All 64000 system covers should be in place and properly attached to the mainframe (all housing screws tight).
- e. Emulator performance verification is a service tool. Minimize its usage consistent with performance assurance.

### NOTE

---

Running the emulator while connected to a target system should produce little additional r.f. noise above that generated by the target system itself.

---



## Appendix **D**

# Emulator Electrical Properties

The emulation equipment, when connected to a target system, will respond similarly to the microprocessor it emulates. The timing of the processor signals at the probe closely approximates the timing of the microprocessor normally inserted in the same plug. Voltage and current requirements for the drive and receive circuitry of the emulator are generally equivalent to LS TTL specifications. The capacitive loading of the emulation probe is equivalent to the LS TTL gate capacitance plus the capacitance of the probe cable, which is approximately 20 pF.

### NOTE

---

The emulation pod presents greater drive capability and slightly greater capacitive loading to the target system than the processor being replaced. Consequently, it is conceivable that a user's system, which operates under emulation, may not operate properly when driven by a microprocessor IC. Noise margins and signal levels in marginally overloaded designs may not cause problems when driven by emulation but may be fatal to system operation under normal microprocessor drive conditions. Be sure that your design allows for the added drive and loading specifications of the 64000 emulation pod.

---



# Index

## a

<ABSFILE> ..... A-1  
 Absolute count ..... 2-13,7-5  
 Absolute file ..... 4-2,5-19,6-8  
 Absolute file format ..... 8-85  
 Accessing existing disc files ..... 8-28  
 <ADDRESS>..... A-1  
 Address .....7-3  
 Address bus width selection .....1-5  
 Address range selection .....1-7  
 <ADR\_LST> ..... A-1  
 Again .....7-2  
 Analysis board..... 1-2,1-5  
 Analysis board installation .....1-5  
 Analysis commands.....7-1  
 Analyzer characteristics ..... 2-12  
 Analyzer status ..... 2-13  
 And function .....7-3  
 Asmb\_sym file .....6-9  
 Assembler symbols file format ..... 8-78

## b

Background memory ..... 2-5,4-8,4-9  
 Background operation .....2-4  
 Background state .....2-7  
 BNC ports..... 7-1,7-6  
 Break ..... 5-2,5-14  
 Break to background memory.....2-5  
 Break conditions.....2-6  
 Bus cable installation .....1-8

## c

Card selection .....4-7  
 Changing a disc file name ..... 8-30  
 Clock selection .....4-8  
 Close display file ..... 8-13

Close printer file.....8-9  
 <CMDFILE> ..... A-1  
 Command delays ..... 5-21  
 Command file ..... 4-4,4-5  
 Command file designation ..... 4-15  
 Command line comment delimiter .....5-1  
 Command word codes ..... 8-22  
 Command word to 8251 ..... 8-40  
 COMN ..... 4-2,6-5,8-87  
 Condition code register ..... 3-1,3-4  
 Configuration .....4-2  
 Configuration questions.....4-6  
 Continue ..... 4-3,4-5  
 Control address ..... 4-13,8-2  
 <COUNT> ..... 7-2,7-5  
 Creating new disc file ..... 8-26

## d

DATA ..... 4-2,6-5,8-87  
 Data .....7-3  
 Data bits switch .....1-5  
 Data jamming .....2-6  
 Deleting disc files ..... 8-30  
 Direct memory access .....3-1  
 Disc file I/O codes ..... 8-33  
 Disc file I/O interface ..... 8-4,8-25  
 Disc file simulated I/O ..... 4-12  
 Disc file types ..... 8-31  
 <DISC#> ..... A-2  
 Display address.....6-3  
 Display commands..... 2-13,6-1  
 Display I/O codes ..... 8-14  
 Display I/O interface..... 8-3,8-11  
 Display simulated I/O ..... 4-12  
 Display techniques..... 8-16  
 Display/list command syntax ..... 6-6,6-7  
 Display/list count ..... 6-13  
 Display/list global\_symbols ..... 6-7,6-8  
 Display/list local\_symbols..... 6-7,6-9

## Index (Cont'd)

Display/list memory ..... 6-7,6-10  
 Display/list memory data ..... 6-1  
 Display/list registers ..... 6-7,6-12  
 Display/list trace ..... 6-7,6-13  
 Don't care ..... 7-3,7-4

### e

Edit ..... 4-5  
 Electrical transparency ..... 2-5  
 Emulate ..... 4-2,4-5  
 Emulation and configuration ..... 4-1  
 Emulation bus ..... 1-8,2-1  
 Emulation command file access ..... 8-25  
 Emulation configuration ..... 2-12  
 Emulation control board ..... 1-2,1-3  
 Emulation control board installation ..... 1-3  
 Emulation controller  
   functional description ..... 2-4  
 Emulation memory ..... 2-11  
 Emulation memory  
   functional description ..... 2-9  
 Emulation pod installation ..... 1-3  
 Emulation pod, 6800 ..... 3-3  
 Emulation pod, 6802 ..... 3-3  
 Emulation probe installation ..... 1-3  
 Emulation probe pin protector ..... 1-4  
 Emulation processor control ..... 2-6  
 Emulation RAM ..... 4-10  
 Emulation ROM ..... 4-10  
 Emulation system data transfers ..... 2-2  
 Emulator clock specification, 6800 ..... 3-6  
 Emulator clock specification, 6802 ..... 3-7  
 Emulator electrical properties ..... D-1  
 Emulator operating modes ..... 2-10  
 Emulator status ..... 3-5  
 Emul\_com ..... 4-2  
 Em6800\_S ..... 4-3,4-4  
 End ..... 5-3

End record format ..... 8-92  
 Ending the mapping session ..... 4-12  
 Error messages ..... B-2  
 Execute ..... 5-4,7-4  
 Exit background ..... 2-7,2-8  
 External clock ..... 2-11  
 External emulation ..... 2-10  
 External symbols record format ..... 8-90

### f

<FILE> ..... A-1  
 File, assembly ..... 4-1  
 File formats, 64000 ..... 8-78  
 File, linking ..... 4-1  
 <FILE NAME> ..... A-2  
 Foreground ..... 2-4,2-7  
 Functional transparency ..... 2-5

### g

<GLOBAL SYMBOL> ..... A-3  
 Global symbol addresses ..... 4-2  
 Global symbols ..... 6-5  
 Global symbols file format ..... 8-82  
 GLOBVAR+ ..... 6-5  
 Guarded memory ..... 4-10

### h

Halt ..... 5-4,5-5  
 Hardware configuration ..... 1-1  
 Host processor bus ..... 2-1,2-3  
 HP-IB bus ..... 2-1,2-3

## Index (Cont'd)

### i

Idle background ..... 2-7,2-8  
 Illegal opcode detection ..... 4-9  
 Illegal opcodes ..... 2-11  
 IMB Master enable..... 7-7  
 Index register..... 3-1,3-4  
 Initialization formats, 8251 ..... 8-56  
 Initialize 8251 ..... 8-39  
 Installation ..... 1-1  
 Interactive commands..... 7-1  
 Interactive measurement  
     configuration ..... 4-14  
 Interactive measurement selection ..... 7-5  
 Interactive measurement  
     specification ..... 4-14  
 Intermodule bus ..... 1-9,2-2,2-3,7-5  
 Internal analysis board..... 7-5  
 Internal emulation ..... 2-10  
 Internal emulation clock ..... 2-10  
 Interrupt vector ..... 3-2  
 I/O bus..... 2-2,2-3

### j

Jam background..... 2-7,2-8

### k

Keyboard I/O interface..... 8-3,8-17  
 Keyboard I/O interface codes ..... 8-20  
 Keyboard output command word..... 8-18  
 Keyboard read request..... 8-17  
 Keyboard read response ..... 8-18

### l

Link\_com file..... 4-2  
 Link\_sym file ..... 4-2,6-8  
 Linker command file access..... 8-25  
 Linker configuration file access ..... 8-25  
 Linker symbols file format ..... 8-80  
 List commands ..... 6-1,6-7  
 Listing file format..... 8-85  
 Load command ..... 4-5,4-10,5-6  
 <LOCAL SYMBOL> ..... A-3  
 Local symbols ..... 2-12,6-7,6-9

### m

Measurement\_system ..... 4-2,4-3  
 Memory accesses ..... 4-8  
 Memory board ..... 1-2,1-7  
 Memory board installation ..... 1-7  
 Memory bus ..... 1-9  
 Memory configuration..... 4-9  
 Memory control board ..... 1-2,1-5,2-9  
 Memory control board installation ..... 1-5  
 Memory mapping ..... 4-9,4-10  
 Memory overlay ..... 4-11  
 Memory size..... 1-6  
 Microprocessor registers ..... 3-1,3-4  
 Microprocessor, 6800 ..... 3-1  
 Microprocessor, 6802 ..... 3-1  
 Mnemonic display/list..... 6-3  
 Modify ..... 5-7  
 Modify configuration ..... 4-6,5-7  
 Modify memory ..... 5-7,5-10  
 Modify memory command ..... 4-10  
 Modify register ..... 5-7,5-12  
 <MODULE> ..... A-3  
 Multi-module system  
     architecture..... 1-2,2-3

## Index (Cont'd)

### n

Nonreal-time ..... 2-9,2-10  
 <NUMBER> ..... A-3  
 Numeric format ..... 2-13  
 Numeric status specification ..... 3-5

### o

Offset  
     addresses .... 6-4,6-10,6-11,6-12,6-13,6-14  
 Open display file ..... 8-11  
 Open printer file ..... 8-8  
 Open RS-232 file ..... 8-38  
 Operating fundamentals ..... 3-1  
 Operating modes, 8251 ..... 8-40  
 Operational command syntax ..... 5-1

### p

Printer I/O codes ..... 8-10  
 Printer I/O interface ..... 8-3,8-8  
 Printer simulated I/O ..... 4-13  
 Processor architecture ..... 3-1  
 PROG ..... 4-2,6-5,8-87  
 Program counter ..... 3-1,3-4  
 Program name file format ..... 8-84

### r

Radio frequency interference ..... C-1  
 Read from 8251 ..... 8-43  
 Real-time ..... 2-9,2-10,2-11,6-5  
 Real-time emulation ..... 2-11  
 Real-time mode selection ..... 2-11,4-8  
 <REAL\_VAL> ..... A-2  
 Register accesses ..... 4-8  
 Register, display/list ..... 6-4

Relative count ..... 2-13,7-5  
 Relocatable file format ..... 8-86  
 Repetitive display ..... 6-2  
 Repetitively ..... 6-10,6-11,7-2  
 Reset ..... 5-13  
 Roll to/write line 18 ..... 8-12  
 RS-232 I/O codes ..... 8-45  
 RS-232 I/O interface ..... 8-5,8-38  
 RS-232 simulated I/O ..... 4-12  
 Run command ..... 4-6,5-14,5-16  
 Running the program ..... 4-6

### s

Select starting line/column ..... 8-13  
 Simulated I/O ..... 8-1  
 Simulated I/O configuration ..... 4-12  
 Simulated I/O error codes ..... 8-67  
 Simulated I/O interfaces,  
     common attributes ..... 8-2  
 Simulated I/O memory allocation ..... 4-12  
 Simulated I/O memory  
     deallocation ..... 4-13  
 Simulated I/O sample programs ..... 8-68  
 Simulated I/O, introduction ..... 8-1  
 Simulated I/O, overview ..... 8-2  
 Single module systems ..... 1-2  
 Softkey status specification ..... 3-6  
 Software interrupt ..... 2-5  
 Source file format ..... 8-85  
 Special considerations, 6802 ..... 3-2  
 Specify ..... 5-16  
 Stack pointer ..... 3-1,3-4  
 <STATE> ..... A-2  
 Static discharge protection ..... 1-3  
 Status ..... 3-5,3-6,7-3  
 Status from 8251 ..... 8-41  
 Status messages ..... B-1  
 Status softkeys ..... 2-13,3-5,3-6  
 Status specification ..... 3-5

## Index (Cont'd)

Status word format, 8251 ..... 8-60  
 Step ..... 5-17  
 Stop\_trace ..... 5-18  
 Storage qualifications ..... 7-4  
 Store ..... 5-19  
 <STRING> ..... A-3  
 Subsystem functional description ..... 2-4  
 Subsystem interfaces ..... 2-4  
 Symbol accesses ..... 4-8  
 Symbols in emulator commands ..... 2-12  
 Syntax shorthand ..... 7-3  
 System bus ..... 2-1, 2-3  
 System bus structures ..... 2-1  
 System command files ..... 5-20

### t

Target system ..... 2-4  
 Target system memory ..... 2-11  
 Theory of operation ..... 2-1  
 Trace ..... 5-16, 7-2, 7-5  
 Trace again ..... 5-4, 7-2  
 Trace command ..... 2-13, 7-2  
 Trace display/list ..... 6-5, 6-7  
 Trace memory ..... 2-13, 5-19, 7-1  
 Trap loop ..... 2-7  
 Trigger ..... 7-1  
 <TRIGGER> ..... 7-2, 7-4

### u

Updating RS-232  
   read/write buffers ..... 8-44  
 User buffer/assembler symbols  
   file packing format ..... 8-80  
 User buffer/linker symbol  
   file packing format ..... 8-85  
 User buffer/relocatable  
   file packing format ..... 8-92  
 User program  
   control address = 00 ..... 8-19  
 User RAM ..... 4-9, 4-10, 4-11  
 User ROM ..... 4-9, 4-10, 4-11  
 <USERID> ..... A-2  
 Using analysis commands ..... 7-4

### v

<VALUE> ..... A-2  
 Variables ..... 2-12

### w

Wait ..... 5-21  
 Write from starting line/column ..... 8-13  
 Write to printer ..... 8-9  
 Write to 8251 ..... 8-41

